uc3m | Universidad Carlos III de Madrid

Informatics Engineering Degree

Bachelor Thesis

"Development of an Augmented Reality musical instrument"

Author: Alejandro Rey López
Tutor/s: Telmo Agustín Zarraonandia Ayo

Leganés, Madrid, Spain
June 2019

Dedication

This piece of work has been a great experience, sometimes a pain in the neck while others incredibly satisfactory. Over these few work-intensive months, and all along the university phase of my life, special people have been by my side, taking care of me and ensuring my welfare.

In the first place, I would like to acknowledge the relentless support of my parents, economically and regarding affection. They express feelings in a strange way, there is no doubt about that, but in contrast to that, they have always trusted on me regardless of what I pursued in life.
Special thanks shall be expressed to my sister, Paula. I undoubtedly cannot conceive a life without her. She is my perfect waste of time, and time is all I got, even though my job is, unfortunately, all about racing the clock.
I would like to express my wholehearted gratitude to Julia, the target of my eyes, the chopper of my burnouts and as of today, my best utterly purposeful would-definitely-do-it-again mistake ever. As you can see, in special occasions, I do not care about English correctness anymore.
I must mention here my primitive workmate, friend and number one backer Mara, an incredible person that is capable of helping others regardless of her own pain. That been said, do not talk to her about politics, just as a recommendation.
In addition to those, I would like to thank my tutor, Telmo, who embraced my initial idea and gave me as many opportunities he could to help me along the way, regarding this dissertation and beyond. I am extremely grateful for your advice and help, as well as your teaching.
Last but not least, I must mention the one person that had the greatest impact in my professional career to date. This project would have never come to life this soon if I had not met José Cabrero during the Informatics Engineering degree.
Problems have been half as tough these last few years just because of him, and I have finally felt good at something after the uncountable number of times I have heard him saying so in every project we have been working on.
He has embraced my will when I did not feel like doing anything at all, and we both have built an amazing coding team via Skype. Being sat all day could have definitely been much more painful otherwise. I appreciate and admire him beyond words can describe.

Development of an Augmented Reality musical instrument

Bachelor Thesis

Alejandro Rey López

Abstract

Nowadays, Augmented Reality and Virtual Reality are concepts of which people are becoming more and more aware of due to their application to the video-game industry (speceially in the case of VR). Such raise is partly due to a decrease in costs of Head Mounted Displays, which are consequently becoming more and more accessible to the public and developers worldwide.
All of these novelties, along with the frenetic development of Information Technologies applied to essentially, all markets; have also made digital artists and manufacturers aware of the never-ending interaction possibilities these paradigms provide and a variety of systems have appeared, which offer innovative creative capabilities.
Due to the personal interest of the author in music and the technologies surrounding its creation by digital means, this document covers the application of the Virtuality-Reality-Continuum (VR and AR) paradigms to the field of interfaces for the musical expression. More precisely, it covers the development of an electronic drumset which integrates Arduino-compatible hardware with a 3D visualisation application (developed based on Unity) to create a complete functioning instrument musical instrument,
The system presented along the document attempts to leverage three-dimensional visual feedback with tangible interaction based on hitting, which is directly translated to sound and visuals in the sound generation application. Furthermore, the present paper provides a notably deep study of multiple technologies and areas that are ultimately applied to the target system itself. Hardware concerns, time requirements, approaches to the creation of NIMEs (New Interfaces for Musical Expression), Virtual Musical Instrument (VMI) design, musical-data transmission protocols (MIDI and OSC) and 3D modelling constitute the fundamental topics discussed along the document.
At the end of this paper, conclusions reflect on the difficulties found along the project, the unfulfilled objectives and all deviations from the initial concept that the project suffered during the development process. Besides, future work paths will be listed and depicted briefly and personal comments will be included as well as humble pieces of advice targeted at readers interested in facing an ambitious project on their own.

Keywords: MIDI Controllers, VMI, NIME, Arduino, Serial2Midi, Unity3D, Drums, 3D Modelling, Blender, AR

Desarrollo de un instrumento musical de Realidad Aumentada

Trabajo Fin de Grado

Resumen

En la actualidad, los conceptos de Realidad Aumentada (AR) y Realidad Virtual (VR) son cada vez más conocidos por la gente de a pie, debido en gran parte a su aplicación al ámbito de los videojuegos, donde el desarollo para dispositivos HMDs está en auge. Esta popularidad se debe en gran parte al abaratamiento de este tipo de dispositivos, los cuales son cada vez más accesibles al público y a los desarrolladores de todo el mundo.
Todas estas novedades sumadas al frenético desarrollo de la industria de IT han llamado la atención de artistas y empresas que han visto en estos paradigmas (VR and AR) una oportunidad para proporcionar nuevas e ilimitadas formas de interacción y creación de arte en alguna de sus formas. Debido al interés personal del autor de este TFG en la música y las tecnologías que posiblitan la creación musical por medios digitales, este documento explora la aplicación de los paradigmas del Virtuality-Reality Continuum de Milgram (AR y VR) al ámbito de las interfaces para la creación musical. Concretamente, este TFG detalla el desarrollo de una batería electrónica, la cual combina una interfaz tangible creada con hardware compatible con Arduino con una aplicación de generación de sonidos y visualización, desarrollada utilizando Unity como base. Este sistema persigue lograr una interacción natural por parte del usuario por medio de integrar el hardware en unas baquetas, las cuales permiten detectar golpes a cualquier tipo de superficie y convierten estos en mensajes MIDI que son utilizados por el sistema generador de sonido para proporcionar feedback al usuario (tanto visual como auditivo); por tanto, este sistema se distingue por abogar por una interacción que permita golpear físicamente objetos (e.g. una cama), mientras que otros sistemas similates basan su modo de interacción en "air-drumming". Además, este sistema busca solventar algunos de los inconvenientes principales asociados a los baterías y su normalmente conflictivo instrumento, como es el caso de las limitaciones de espacio, la falta de flexibilidad en cuanto a los sonidos que pueden ser generados y el elevado coste del equipo.
Por otro lado, este documento pormenoriza diversos aspectos relacionados con el sistema descrito en cuestión, proporcionando al lector una completa panorámica de sistemas similares al propuesto. Asimismo, se describen los aspectos más importantes en relación al desarrollo del TFG, como es el caso de protocolos de transmisión de información musical (MIDI y OSC), algoritmos de control, guías de diseño para interfaces de creación musical (NIMEs) y modelado 3D. Se incluye un íntegro proceso de Ingeniería de Software para mantener la formalidad y tratar de garantizar un desarrollo más organizado y se discute la metodología utilizada para este proceso. Por último, este documento reflexiona sobre las dificultades encontradas, se enumeran posibilidades de Trabajo Futuro y se finaliza con algunas conclusiones personales derivadas de este trabajo de investigación.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Preamble

One of the handicaps for musicians all over the world is space.
Musical instruments usually require a decent amount storage space and additional cares so that their sound quality remains intact. Therefore, owning an instrument often either implies expensive rents for assumable wide flats or requires the owner to rent additional room for them outside its own house (r.g. rehearsal rooms).
In particular, drummers and percussionists are specially sensitive to the aforementioned problem, as the dimensions of a conventional drum-set are very space-demanding; even for a basic drum setup, such as the commonly referred to as "**four piece drum-kit**" (which is a basic but versatile kit composed by snare-drum, hi-hat, bass drum, a floor tom and two cymbals [usually ride and crash]). Such simple drum configuration, including a throne to sit on and play can take about 7' wide x 5' deep (2,1336 m x 1,524 m) optimistically.

In addition to the trivially spotted issue related to space, learning to play drums involves many other concerns and headaches. **Percussion is loud** and this imposes various limitations on practising periods, which need to be adapted to avoid problems with your neighbours if you cannot afford a sound-proof studio room.
By now, I have covered the ground for those musicians who already own a drum-set, either an acoustic drumset or an electronic one and have actual access to those.
However, this is not always the case; being the author to this paper an example of such misfortune. The author himself owns a drumset that he has not played for ages because of the constraints imposed by living in a small flat in Madrid, as he began the Informatics Engineering degree. And this was the main reason for this project to be devised in the first place.

For those that may be looking forward to joining the drum world, there are, as evidenced here, many obstacles that will potentially discourage you from bracing the pair of sticks you bought, since there are even more concerns to consider before

getting started.
Drums are likely to make you bankrupt easily; both drum-set alternatives (acoustic or electronic) are highly expensive, as the price of any drum accessory you can imagine easily surpasses one hundred euros. In particular, a basic cost-quality balanced acoustic drum-set is in the region of 450 euros, equipped with very poor quality cymbals, which by the way, generally constitute the most expensive piece of any drum-set. A single cymbal, which after all is piece of metal you hit fairly frequently, can cost more than 300 euros. Consequently, breaking a cymbal is for certain something you do not want to do, and if you are just starting out and you are not confident about whether you may eventually get tired of drums, you may look into other options.
Regarding electronic alternatives, disparate basic sets are in the region of 400 €, which have an acceptable feel and are suitable for beginners too.
In any case, the reader may have noticed it is still a lot of money and portability is limited for both kinds of instruments. For those that inhabit small rooms such as the author of this dissertation, the use of practice pads is a commonality, since these products can be placed in barely any surface and make practice easier on the go.
As an illustration, people often place practice pads on their bed, and complement the hand-hitting pad with a a pedal pad, so that practice can get as realistic as possible. However, there is still a problem regarding the limited amount of sounds you can get from practice pads; since they only allow you to practice rudiments, with no more similarities with regard to real drumming than hitting a chair. In addition, practising limb independence of movement becomes complex due to the fact that understanding what is going on with your hands and feet without a direct mapping with sounds is cognitively overloading. The practice experience easily turns into frustration or boredom, specially if you rarely sit in front of a real drum-kit and prove your hours of practice have been worth.
All that been said, recording has not been mentioned yet; this matter is threatening as well. Recording your drums involves even more space and money; either you get a quite expensive electronic drum-set that you can connect to a DAW[1] or you will be spending hundreds of euros in a set of microphones in order to capture the sound of your drums with a decent quality. This is an outstanding source of frustration for the author himself, inasmuch as he would come up with lots of musical ideas but ,unfortunately, never owned the gear to record them properly.
Of course, there are manifold VST (Virtual Studio Technologies) such as *Addictive Drums*[1] that make a really good job simulating drums and offering a great amount of pre-recorded beats that you can modify yourself or control using a MIDI keyboard controller, assuming you own a MIDI keyboard controller in the first place. Purchasing a MIDI keyboard controller implies yet another outlay, even though they are more affordable than electronic drums, but the cost of a home drum set-up for recording seems like growing exponentially.
On top of that, many musicians would agree that it is not pleasant to record drums by hitting keyboard keys that provide no guide for playing, specially once you are used to sticks and foot coordination that are skills you have developed over the years with an actual drum-kit in front of your eyes. In addition, key pressing precision is not comparable to seamless playing and the recording process for keyboard-shaped

[1]DAW: stands for Digital Audio Workstation, a program to record and mix music tracks.

MIDI controllers often gets so tedious that in order to get a result that sounds far from robotic you may need to spend a whole day editing MIDI tracks.

Driven by everyday issues regarding drummers, the whole point of this dissertation is to present a complete drum-set that enables a musician to overcome many problems regarding sound-proving, costs associated to recording equipment and the space limitation without compromising the "hit feeling" without the need of any kind of real drum-set pieces but a pair of custom drumsticks and the system which will be detailed through this paper. ***DrumVR*** has been conceived to allow a natural drumming experience, whose scope goes from practising to recording, powering the otherwise easily tiring experience by means of the tools offered by the Augmented Reality paradigm (visual feedback, holographic visualisation, 3D sound...etc) and the help of physical interfaces.
Having a visual guide for practising is known to be very helpful, as it allows for a complete training without requiring a musician to carry an actual drum-set with them wherever they go. In addition to visual feedback, ***DrumVR*** is to be providing sound effects so that the user can focus in the drumming experience and the possibilities, mostly regarding flexibility, enabled by musical instrument virtualization.
It is important to note that the system has been devised as a proof-of-concept system toward which further expansion is to likely be carried out in even after the completion of the dissertation itself. The author believes that building a market-ready product would be too challenging for only one person, considering the relatively tiny amount of know-how a Computer Science undergraduate has beyond the basics on many different inter-related topics. Therefore, the objectives presented in the following section constitute a set of goals that are, ideally, to be implemented, yet may end up going far beyond feasibility, or may be left out as future work.
In order to the development, the hardware of the system will be based on the Arduino hardware, which is an extensively used set of prototyping technologies; alternatively, the 3D visualisation environment will be created with the tools that the Unity3D framework provides. The instrument virtualisation system will be used to provide auditory and visual feedback to the user in addition to allowing for configuration, so that the basics feedback loop present within a real-world musical instrument (e.g. a drumset) can be effectively replicated in virtual space.

1.2 Objectives

This sections aim to enumerate and briefly enlighten the main higher level milestones which are to be approached in the following sections for completion.

As it was mentioned in the Preamble, this document aims to build a prototype of a musical instrument with custom assembled and programmed hardware that allows for a user to trigger a sound on the computer and provides means of visualisation while interaction is enabled. However, I believe it makes more sense to modularize the objectives further so that the chief purpose of this document is stated in a more concise way from the very beginning; avoiding extra drifting due to a vague general

description about what is to be achieved.

- **Objective 1: Creation of a physical interface for musical generation**: a tangible interface that resembles that of real drum playing and effectively transmits and processes musical interaction messages is to be designed. The idea is that it is based upon a set of drumsticks that users can use to hit a random surface and generate sounds in response.
- **Objective 2: Creating a 3D visualisation application that represents the drumset in virtual space and enables triggering sounds associated to the different present components:** a software system that processes input obtained from the physical interface mentioned earlier is to be create. This application is the one whose responsibility is bound to provide some kind of response to the user input, that is, it shall at least provide sonic and visual feedback.

Even though the previous objectives effectively delimit the scope of the project, there are extra considerations that can be heeded as well as a must for the success and utter usefulness of the system described herein.:

- **Performance and Usability**: due to the capital time requirements that musical performance imposes, delay shall be minimised and covered in depth. A system that does not allow real-time interaction constitutes a completely unusable and annoying musical instrument whose usefulness is largely restricted; in other words, it serves as a mere demonstration of the practical potential a well-performing system similar to the presented one would have. Such lack of practical usability is not desirable and will be avoided to the extent possible. In this sense, it is known that percussion musicians are particularly sensitive to sonic delay and thus, performance is even more constrained due to this fact. Additionally, usability is a goal to be achieved by means of, among other things, flexibility regarding the range of sounds that can be generated with the system, in addition to portability, which must be taken seriously.
- **Aesthetic and ergonomic design**: An extra effort shall be made towards providing both a good-looking and functional whole, since it is intended to be used by musicians, which are accustomed to aesthetically appealing real-world instruments and hence, beauty becomes part of the experience. Moreover, it is an important aspect to consider ergonomics and musical instrument design guidelines when it comes to build or system, regardless of the fact that the system to be build is a proof-of-concept.

1.3 Document Structure

For the sake of clarity, the present document is divided into chapters, all of which are briefly summarised next, to provide with an overview of what the reader can expect from this work.

- Chapter 1, *Introduction*, presents an overview of the contents of the document, including the main issues that motivated the project, the main objectives that are to be achieved and a summary of the structure of the dissertation.

- Chapter 2, *State of the Art*, provides with a brief panoramic of the history of synthesizers, discusses the MIDI protocol 1.0 in some detail and elaborates on the most notable Virtual Musical instruments and Virtual Reality Musical instruments created to date by researchers all over the world. All of this information is then used to assess the approaches that can be used for this project to come to life in a comprehensive digest (see Section 2.4 ,which constitutes a feasibility study on its own regarding the creation of the system.

- Chapter 3, *Analysis of the Problem*, defines the scope of the project in formal terms, building on the main idea and progressively making it more and more precise. Eventually, it depicts the definition of requirements into two abstraction levels, User Requirements and System Requirements.

- Chapter 4, *Design of the solution*, starts off by evaluating the alternatives that were looked into as possible solutions for the requirements detailed before. From those approaches , only some are chosen to proceed with the architectural design and implementation details, which are provided immediately after.

- Chapter 5, *Evaluation*, details the testing plans that have been elaborated to assure the created system complies with the requirements defined during the elicitation process.

- Chapter 6, *Project Plan*, elaborates on the methodology used for the project, including the foundations upon which that selection was made as opposed to others. It also plots the formal planning performed at the very beginning of the project and compares it to the actual project schedule, so that such data can be used as a reference for similar projects.

- Chapter 7, *Socio-Economic Environment*, summarises the monetary and time resources that the completion of the project involved, along with a discussion on the impact the project may have in this matter.

- Chapter 8, *Legal Framework*, debates about the laws that concern the project and the different licences under which third-party software used extensively herein are released.

- Chapter 9, *Conclusions*, condenses the main outcomes from the project and provides with future work proposals as well as some personal comments regarding the experience obtained from the completion of this project.

1.4 Background concepts

This section aims to provide a quick overview of concepts that may be used widely along this dissertation. It is targeted at providing a clear but concise knowledge

base to readers that are not familiar with the Computer Science or musical jargon, so that a better understanding of this document can be achieved.

- **Virtual Reality, Augmented Reality and Mixed reality:** both concepts of *Virtual Reality* and *Augmented reality* are tightly related and it is useful to present them together so that we can understand their differences in an appropriate context.
  The RV Continuum was a concept introduced by Milgram et al. in an attempt to provide a universal definition for these a priori abstract concepts [2]. The distinction between a "pure" Real Environment and a "pure" Virtual Environment lays in the nature of the objects that compose these environments; that is, a real environment contains solely real objects (those that exist in the real world); while a Virtual environment is made of a completely synthetic world, which may or may not mimic the real-world appearance of objects, but is essentially non-real, where physical laws characterising a real environment no longer hold.
  Everything that lays between those two "pure" environments is called ***Mixed***

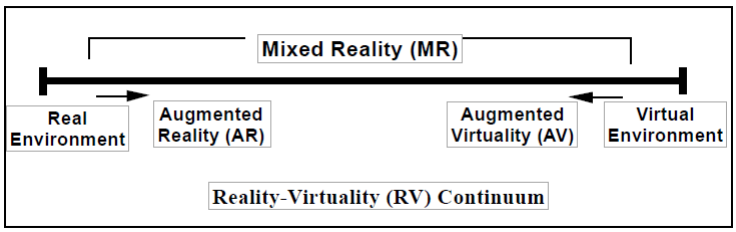


Figure 1.1: Schema of the Reality-Virtuality Continuum [2]

Reality (also referred as XR). Therefore, a Mixed reality application or display is simply overlaying real-world objects or elements with virtual or CG (Computer generated) ones. According to the importance of the physical and virtual objects in the environment of the application respectively (referred by Milgram as *substract*) we may further classify a display as VR or AR.
In a broad sense, an ***Augmented reality*** application would be one which blends virtual objects into reality; that is, the real-world is added some virtual objects to it to expand its reach and the capabilities offered by it. ***Augmented Virtuality (AV)***, on the other hand, is based upon a synthetic world to which real world's elements are added (e.g. hand control of virtual objects).
Virtual Reality, on the other side, is in the right-hand corner of the Continuum, so it corresponds to a pure virtual world representation. These definitions are oversimplified and discussed in more detail by Milgram et al. They distinguish three dimensions that allow a better classification of devices and applications within the Mixed Reality Continuum, which are briefly summarised next:

  - **Reality**: degree of virtuality or reality in the sense that the displayed objects are primarily real or computer-generated.

  - **Immersion**: level of success in making the user actively feel as part of the environment it is being displayed.
  - **Directness**: level of naturality with which users visualise the mixed reality world (i.e. objects may be directly visible through optical HMDs or processed prior to display).

- **Reproduction fidelity (RF)**: degree of accuracy achieved by a system according to either the quality of the images it displays or the *immersion* level offered by it with respect to providing practical usefulness. Thus, fidelity can refer to how realistic imagery looks in both temporal and spatial resolutions; the goal is to make the user feel as there is no screen between it and the reality being watched, a concept which is usually referred to as "unmediated reality". Immersion, on the other hand, relates to the degree to which the user can feel as within the environment; in other words, immersion does not necessarily depend on realistic looking graphics, but it is a concept based on a combination of how well the displayed objects enable the sensation of "being there".

- **Application Data or "useful data"**: in relation to data transmission protocols, we refer to application data as the information that concerns an specific application and has to be conveyed between two ends. This data often requires encapsulation in order to be transmitted between ends that are not just random, but instead have some way to be identified. For example, in the case of communication over the network, a set of protocols are used to allow the information to be sent towards a specific computer within the network; for that, more data than the message to be transmitted (e.g. a webpage) needs to be transmitted along with the message itself [3].

- **Tuple**: and n-tuple, in mathematics and computer science refers to a list of n ordered elements. These are often parenthetical expressions, even though other delimiters such as square brackets "[]". An example of a 2-tuple could be (1,3); thus, tuples can be used to describe other mathematical objects (e.g. vectors) [4]. In this dissertation, we will refer to tuples as data values that belong together (as it is the case of time-stamps and a MIDI message, for instance).

- **New Interfaces for Musical Expression (NIME)**: is an international conference devoted to scientific research that plays a role in musical expression and artistic performance. Within this important event, concerts, workshops and talks regarding innovative approaches to musical interfaces take place. Mutiple research areas converge in these conferences, including muscal education, protocol design, real-time computer systems...etc [5].

- **Digital Audio Workstation (DAW)**: it is a device or software applcation that is used for recording, editing and producing audio files. Among the most notable commercial DAWs we can find Ableton Live, Adobe Audition and Cubase [6].

- **Virtual Studio Technology (VST)**: it is a software interface developed by Steinberg (an e-instruments brand), which is used to interconnect synthesizers and effect plugins to different recording systems (such as DAWs). Generally,

a VST is used to refer to an audio processing plugin that is used either alone or in conjunction with other sound processing software, referred to as VST host; at the end of the day, VST these are just libraries which provide audio processing functionality [7].

- **Tangible Interface** (TUI): it is a User Interface with which the user interacts with the digital information through physical objects. These leverage the the human ability to grasp and manipulate physical objects and materials [8].

- **Proprioception**: it a sense that drives the ability to control one's body and its movement by figuring out the relative position of the contiguous body parts. The proprioception sense is responsible for equilibrium and coordination among other things [9].

Chapter 2

State of the Art

There have been many attempts to apply the relatively new Virtual Reality and Augmented Reality paradigms to the music environment and the creation of highly flexible musical instruments. However, the concept of computer-aided musical composition has been around for centuries, and research has settled the very basics for the upcoming generations of music and computer-science enthusiasts to keep exploring the uncountable possibilities that computers can provide traditional music with.

Along this section, the reader will be presented with a brief overview of the history of electronic musical instruments, more precisely synthesizers (Section 2.1).
Afterwards, and following the mention to the revolutionary MIDI protocol in historical terms, the reader will be guided through a comprehensive explanation of the MIDI protocol itself, thus, accounting for its selection, yet considering its limitations (see Section 2.2)
Next,in Section 2.3, an analysis of the recent efforts made to integrate technologies belonging to the *Milgram's Reality-Virtuality Continuum* [2] with the creation of music will be discussed.
As a summary, an stress will be made to clearly define the virtues and disadvantages presented by most of the proposed solutions (that is to say, virtual and augmented instruments developed to date), and general conclusions regarding the discussed material will be depicted in Section 2.4.

2.1 History of Synthesizers and the origin of MIDI

In order to find the very first footprint of electric musical instrument, we must look back not years but centuries, back to the 18th century.
The first electric-powered musical instrument dates from 1759, when Jesuit priest Jean-Baptiste Delaborde (France) built the Clavecine Électrique, also known as the ***'Electric Harpsichord'***.Instead of being a stringed instrument, it looked more like a carillon type keyboard, which used a static electrical charge (supplied by an early form of capacitor) to vibrate metal bells. The early instrument allowed to

generate a series of sustained notes, similarly to an organ [10].

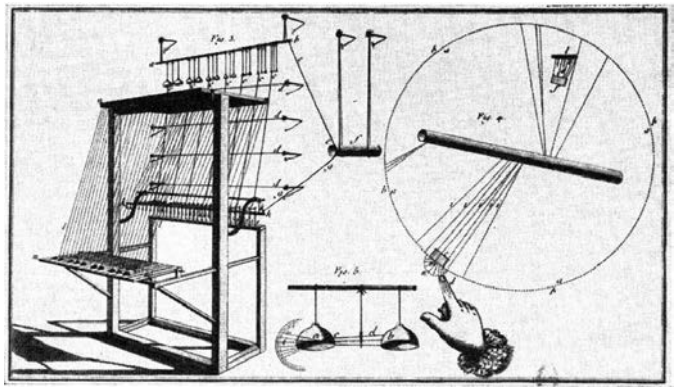


Figure 2.1: Sketch of the *Electric Harpsichord* ; source [10]

Later, entered the nineteenth century, Elisha Gray created the first purely electrical sound generator, the ***"Musical Telegraph"*** (see Figure 2.2), which was inspired by Gray witnessing his nephew playing with around with his equipment. The child had connected one end of a battery to himself and the other to a bathtub; by rubbing his hand on the bathtub's surface he created an audible humming tone proportional to the electric current as input.The idea of the inventor was allowing to send multiple telegraphic messages encoded as different pitches simultaneously over the same line, so that these could be decoded at the receiving end. Thus, this device was capable of remotely transmitting the generated music when connected through telephone cable along more than 200 miles [11].

Later discoveries were based on Gray's device, such as the ***Telharmonium***, in-

Figure 2.2: Gray's Musical Telegraph; source [11]

vented by Thaddeus Cahill in 1895 and considered the first actual synthesizer; it was conceived by the inventor as an *" "Apparatus for Generating and Distributing Music Electrically"* and designed as to enable the generation of perfect pitches (notes), whose sound could be altered to match the timbre and characteristics of other instruments as well; making it a versatile instrument [12].

The popular ***Theremin*** appeared in 1922, which exploited the Heterodyning effect, discovered shortly before. This phenomena allowed to generate a monophonic sound

whose pitch could be controlled manually by the substraction of similar waves, generating a wide range of audible frequencies.
Rapid advances in subsequent decades regarding analog signal manipulation conveyed to the assembly-line production of the first synths that were commercialised, such as the ***Novachord*** (see Figure 2.3), manufactured by the Hammond Organ Co in the USA from 1939 to 1942, which introduced simple pressure sensitive system that allowed control over the attack and timbre of the notes, and which was played using an interface like a common piano [13].

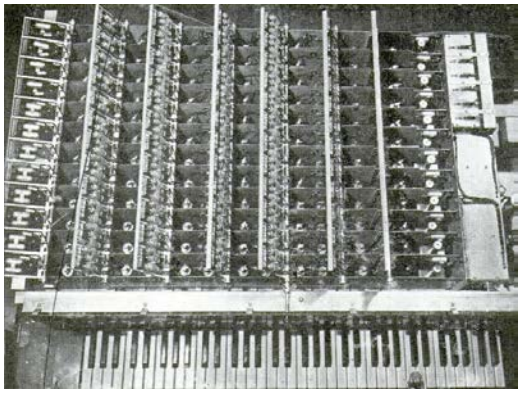


Figure 2.3: Hammond Novachord; source [13]

Time brought novel capabilities to basic electronic instruments, which allowed the development of the ***Electronic Sackbut*** (see Figure 2.4) by Canadian Hugh Le Caine in the late 40s. This was the first musical instrument enabling real-time control of sound's volume, tone and timbre.

Figure 2.4: Electronic Sackbut available at: [14]

After World War II, the first recording studios appeared, and with them, the modular synthesizers arised, which were also programmable (the first one was the Mark II Sound Synthesizer, which was the size of a room and interconnected many different pieces); these new modular devices allowed for an increased flexibility of synths to date.
Around the 60s, with the newly emerging transistor technology, the first **portable modular synths** entered the stage (i.e. Moog [1964]), although they did not become commercial until the development of really tiny solid-state components (the 80s),

which allowed synthesizers to become self-contained and truly portable instruments. By then, analog voltages were controlling how sound behaved, regardless of the brand producing electronic instruments. Nevertheless, compatibility was a massive issue, as each manufacturer was implementing its own variant of an analog interface for synthesizer modules. This was a problem in the sense that a performer could not know whether the assembled modules, when connected, would result in the sound obtained from an equivalent module from a different brand, and it was hard to build up complex heterogeneous systems that could be controlled by a master keyboard, for example.

To solve this mess, Bryan Bell (a sound engineer) created a common interface that enabled the connection of heterogeneous modules and their configuration to achieve desired sounds in an automated way, constituting the first step for interface homogenisation in the musical industry.
By the end of the 70's, many synthesizers implemented a digital microprocessor, controlling the analogue signals that manipulated sound, and implementing features such as automatic tuning, pattern generation and sound storage.
To allow interconnection of their synthesizers, Roland introduced a technology called DCB (Digital Control Bus) to enable interconnection of their products, along with a special connector, the DIN-SYNC, which was designed so that it was impossible to mis-connect devices.

In the early 1980's, in an attempt to standardise a digital protocol for commu-

(a) MIDI ports and cables

(b) MIDI association Logo

nicating between synthesizer modules from different manufacturers, leaders of the digital instrument industry co-developed **MIDI** (Musical Instrument Digital Interface), which was introduced at the 1983 NAMM trade show [15]. MIDI lays the most basic foundation of this project, and will be discussed briefly next.

2.2 Overview of the MIDI protocol

This section covers the fundamentals of MIDI, which is the most widely used communication protocol for electronic musical instruments. Note that other communication protocols exist, as we will discuss later in Section 2.3; but this one was chosen to be used in development because of its ubiquitously.

2.2.1 What is MIDI? Fundamentals

MIDI is a standard communication protocol; this means it is simply a clearly defined step-by-step communication procedure between two machines, or in other words, a set of steps for two machines to make sense of the messages being exchanged between them at a given point in time.
In particular, MIDI is a *Serial Communication protocol*, precisely meaning that communication can be performed with only two cables (or even one only), one controlling the timing of the data exchanges (CLK) and the other being used for the actual data exchange (IN/OUT); therefore, the same bus is used to transmit data from one end to the other, so they must agree upon the use of their common link to transmit information.
In fact, there is a special serial communication protocol which is **asynchronous**, meaning that no clock signal is used to control the timing of the data exchanges, allowing us to use a single cable for the whole transmission process. This is the real protocol used by the MIDI specification as lower-level transmission control mechanism. From now on, we will be focusing on the insights of this asynchronous variety, and we will describe how it works roughly.

Let's say we want to transmit the word "Hi" from S1 to S2. We could be using MIDI for that, or any other serial communication protocol. However, it is easy to spot a problem with uncontrolled data transmission, data from both ends may easily collide and get tangled. Owing to that problem, there are some mechanisms targeted at avoiding it:

- **Baud rate**: the clock bus or independent end configuration is the main synchronisation mechanism, it specifies how fast data is to be sent over the serial data bus. The units of the baud rate are usually bps (bits per second even though they are often referred to as bauds), so if you raise those units to the power of minus one, you can easily obtain the time it takes for the protocol to transmit a single bit based solely in the send time (neglecting the time to transmit over the serial bus itself).
  Example: 9600 bauds (bps)

$$\frac{1}{9600bps} = 1,041666667^{-4}s = 0{,}10467 \text{ ms} = 104\mu s$$

  The "useful" data each frame carries is often 8 bits, and then, we have special bits (namely Start and stop bits, discussed next) which tell the other end when data has been already transmitted, so that the other machine can read the line. Then, the upper layer protocol may impose more rules and reduce the amount of bits that are usable or fixed, as we will see with MIDI.

- **Synchronisation bits**: System 1 and System 2 (namely S1 and S2 in Figure 2.6) must know when one has finished transmitting, so that the other can read whatever zeroes and ones lay in the receiving bus. For knowing so, there are

2 or 3 special bits that are added to the app data notifying whether the data chunk being sent is complete or not.
The *Start bit* is always a zero, notifying the other end that it is going to start the transmission of a packet (the bus state changes thus from one to zero meaning "I start communicating"). When the data transmission is to be finished because the packet has been fully sent, one *Stop Bit* is sent (or occasionally two), with value 1, so that the other end can now process the message accordingly (extracting the application information contained in the packet; i.e. "Hi").

- **Parity bits**: parity is a very rudimentary way of low-level error checking, which consists of adding up the data bytes and storing as a bit the evenness of the sum. Often, if the data byte contains an odd number of ones, the parity bit gets set to 1; to make the whole thing even. In the case the data byte contains an already even number of ones, then, the parity bit is zero.
  This allows for discarding packets with tiny errors, but it is an very unreliable mechanism overall, so it is not used very frequently. Neither is it used in MIDI, so it doesn't appear in the examples, but it was mentioned for completeness.

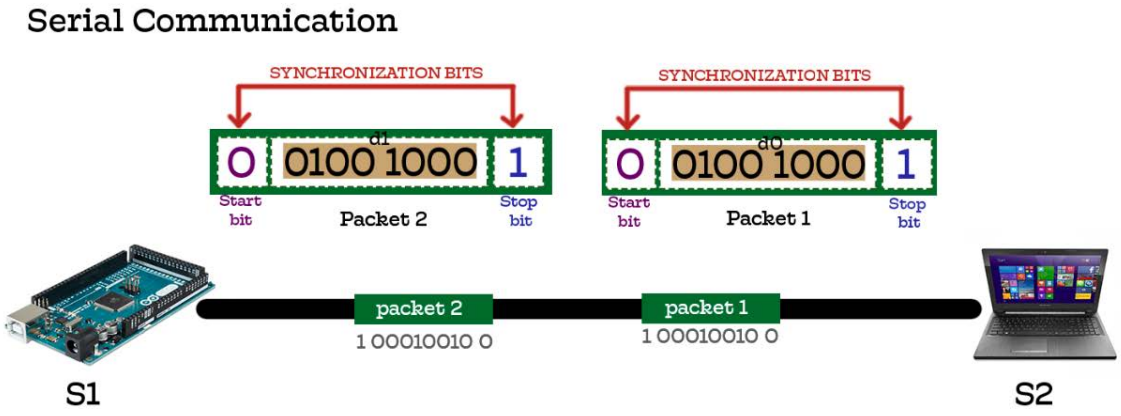


Figure 2.6: Serial Communication example; transmitting "Hi"

Referring back to our example, Figure 2.7 shows the required steps to be performed so that application data can be transmitted, suffering different transformations and encapsulations along the way to allow transmission (often referred as *Marshalling*[1] or *Serialization*):

If we aim to transmit "Hi" using a Serial protocol, we will need to convert the characters independently into its encoded equivalent, and then send those in two separate packets, due to the 8-bit "useful data" length limitations imposed by data framing. The Baud rate must be adjusted in the same way in both ends (S1 and S2) for the communication to work, otherwise, you will notice communication errors all the time, since no end will be able to understand what the other is trying to express (they would be de-synchronised).
In the example, when aiming to transmit "Hi", we would need more bits other than

[1] `https://en.wikipedia.org/wiki/Marshalling_(computer_science)`

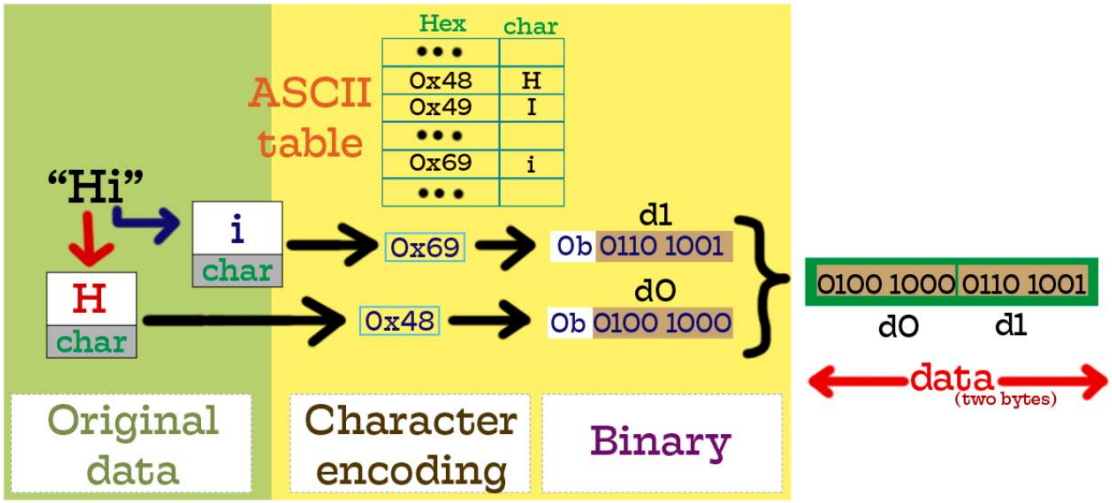


Figure 2.7: Data to ASCII to binary encoding

those from the content itself, which are start and stop bits. Once we converted the expression "Hi" into numeric values that can be understood by a computer (binary encoding), the resulting chains of zeroes and ones will be sent over the Serial bus. Note that d0 and d1 in Figure 2.6 are those obtained from the transformation pipeline described later in Figure 2.7.

2.2.2 How does Serial relate to MIDI?

MIDI is focused in the transmission of the data necessary to generate sounds, not vain greetings like we saw in the example. MIDI is used to transmit whether live sounds (which are triggered by a musician on an electronic keyboard) or sounds stored in files, which gather information of the sounds constituting a song (tempo, notes, duration of each note, instruments....) so that it can be played back.
MIDI messages comprise a set of application data bytes sent over Serial, which are logically arranged in order to make sense of their meaning. The different types and lengths of these messages will be described shortly .
There is a possibility that you have heard about **Standard MIDI Files**; these are related to the MIDI protocol because they are created by storing a set of events (MIDI message receipt), which appear within the file as a timestamp-MIDI message tuple. Each tuple simply specifies which sound to trigger (encoded in the MIDI message) and when to generate it when played back, which is specified by the time component of the message. Moreover, files record the time to wait before the next sound is generated, so that the musical "performance" can be defined solely as a set of events triggered sequentially with silence among different events.
With these concatenated tuples, we are able to play back a MIDI file that maintains tempo and sound characteristics recorded in the first place. In this dissertation we are not largely interested into how MIDI files work, but instead, we are concerned about live MIDI sound generation, so we will stop the discussion about this topic here. You can take a look at [15] for a more detailed discussion on the encoding and the limitations of Standard MIDI file encoding.

Even though there is plenty of room for other applications falling within the MIDI specification, MIDI was not conceived for data transmissions out of the scope of music, and other applications may neither fit nor afford the limitations of this protocol. Next, we will dive into the details of MIDI messages and general applications of each one.

2.2.2.1 MIDI Sound generation and MIDI messages

In order to generate sounds, the MIDI protocol provides with different types of messages, with their characteristic structure and purpose. All those are transmitted between machines through a Serial interface, as discussed previously.
MIDI uses a baud rate of **31,250 bps** as standard, which means that the maximum amount of data that can be sent per second is 3,125 bytes per second (since 8-bits of data require 2 extra bits (start ad stop bits), so 10 bits need to be transmitted for the shortest whole MIDI case scenario).

A **MIDI message** is no more than a set of bytes, chained zeroes and ones grouped into 8 digit-chunks. Each type of message has a specific length, depending of its purpose. Any MIDI message is made out of two types of bytes (chunks of 8 bits): status bytes and data bytes:

- **Status bytes**: identify the type of MIDI command being triggered. All status bytes begin with a one; that is, their Most Significant Bit (MSB) is 1. Status bytes can be subdivided into two parts of the same length, both 4 bytes, called *nybbles*.
  The most significant nybble (upper four bytes) contains information regarding the type of command to execute, whereas the second specifies the MIDI channel to which the command applies to. As the MSB is fixed, there are only eight possible command types, therefore, eight categories of messages.
  On the other hand, regarding channels, we are allowed to use up to 16 as it is possible to utilise the LS nybble bits to all its extent (4 bits; $2^4 = 16$) .
- **Data bytes:** if the first bit (MSB) of a received byte is a zero, then, it solely contains data, which will be associated with the last received status byte, which often will be build up together a complete MIDI message. The meaning of each data byte is determined by the category the previously received status byte fell into.

Knowing these two types of bytes, we may finally discuss the variety of MIDI messages that can be built. Essentially, there are 8 types of messages namely: ***Note Off***, ***Note On***, ***Polyphonic Pressure***, ***Control change***, ***Program change***, ***Channel Pressure***, ***Pitch Bend*** and ***System*** messages.

- **Note On**: This command allows for specifying a note to be played back, and

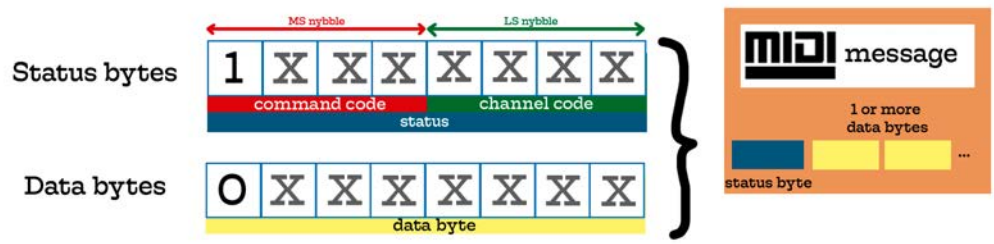


Figure 2.8: MIDI byte types

requires information that is encoded in two subsequent data bytes: the note number and a measured velocity depending on how hard we hit or how loud we want the note to sound like. If we think of a usual piano-looking MIDI controller, each key would be associated with a specific Note number, and often, a pressure sensitive piece of the key is the responsible for telling the computer the velocity value.
We can direct the command to any of the 16 MIDI channels by telling to which it applies in the least significant nybble of the status byte.
As an example of this message, say we aim to create an E4. To do so, we would look for the number between 0 and 127 that is associated to the desired note (in this case E4=64) and then we would specify a velocity to taste. Velocity values are supposed to have correspondence in terms of classical music theory, however, different instruments handle velocity values in a variety of ways, as mentioned in [16].
If we do the math, we can compute time it takes to transmit a Note On complete message (One status byte and 2 data bytes):

$$3bytes/3.125bytes/s = 0.00096s = \mathbf{0.96\ ms} \tag{2.1}$$

- **Note Off**: Contrary to the previous type of message, this one is used to silence a given active note which was previously triggered by a Note On message. The difference in the structure with regards to note on messages is non-existent; these two messages look exactly the same but the command code for them is different.

Note On and Note Off messages are closely related, so in fact, you can use a Note On message as an "*implicit Note Off*" message by simply specifying on it a velocity value of 0; the effect will be the exact same one we would expect from a Note Off message targeted at particular note number.
In addition to this eventuality it is also possible to save bandwidth by using what is called ***Running status***, which consists in concatenating many Note On/Note Off information after a single status byte, so that if we aim to send every note towards a unique channel, we save time by only advertising we will be sending a stream of Note On/Note Off data bytes next.
In other words, unless there is a new status byte arriving to the receiving end, all the data messages will be processed as belonging to the formerly specified

channel and data bytes will be split in groups of 2 data bytes as if each one was preceded by its own status byte (constituting a standard MIDI message). You can find more precise details in Figure 2.9

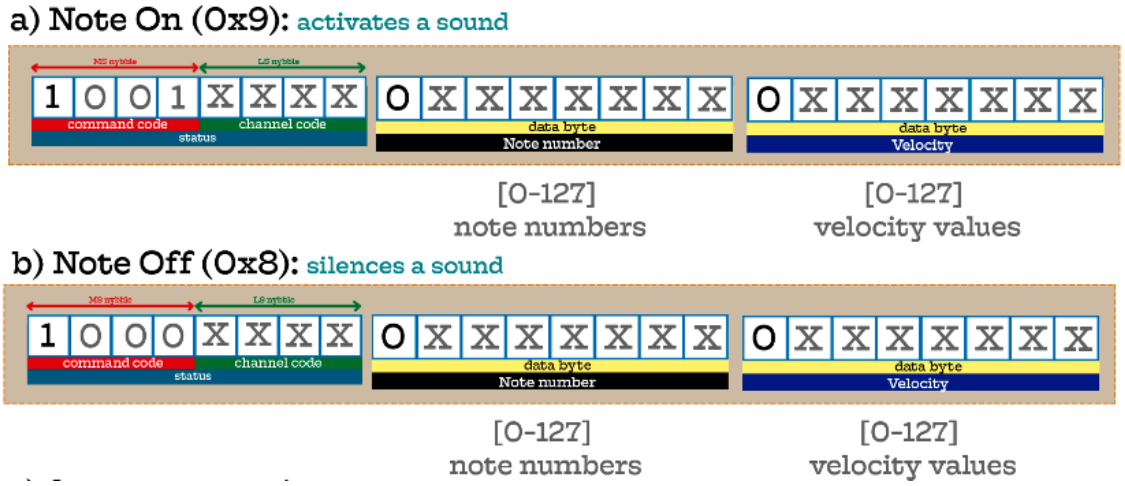


Figure 2.9: *Note On* and *Note Off* messages

- **System Real Time**: These messages are easily recognised, as are those that have the most significant nybble of the status byte set to **0xF** and the first bit (msb) of the second status nybble set (that is, set to 1). These messages are used for synchronisation purposes, often to keep share the tempo among sequencers or to start/stop the recording. System real time messages are only one byte long, that is, they comprise only the status byte. Within these, the channel is not specified, instead, those bits are used to tell which is the command itself within the category of System Real time messages. As they do not have data bytes, we can use them without interrupting "Running Status".

- **Polyphonic Pressure**: To provide an instrument with more expressiveness, some MIDI controllers incorporate a feature colloquially referred to as *Aftertouch*. One of the kinds of Aftertouch is the so called polyphonic aftertouch, in which every key is capable of sending its own independent pressure information.
  A polyphonic pressure message is composed of a status byte and two data bytes. The status identifying a message as Polyphonic Pressure is 0xA, and as in most message types, the ls nybble contains information regarding the channel receiving the message. The first data byte specifies the key number associated to the pressure value to be controlled, whereas the second value is the value for the pressure exerted towards the pressed key at a specific moment (similarly to velocity).

- **Channel pressure**: as it is pretty expensive to include a pressure sensor underneath each key of a MIDI controller, a cheap way to add expressiveness to an instrument is to provide with a overall pressure measure. Channel pressure messages gather information regarding the pressure of the notes being played for a specific channel. Channel pressure messages are identified by the com-

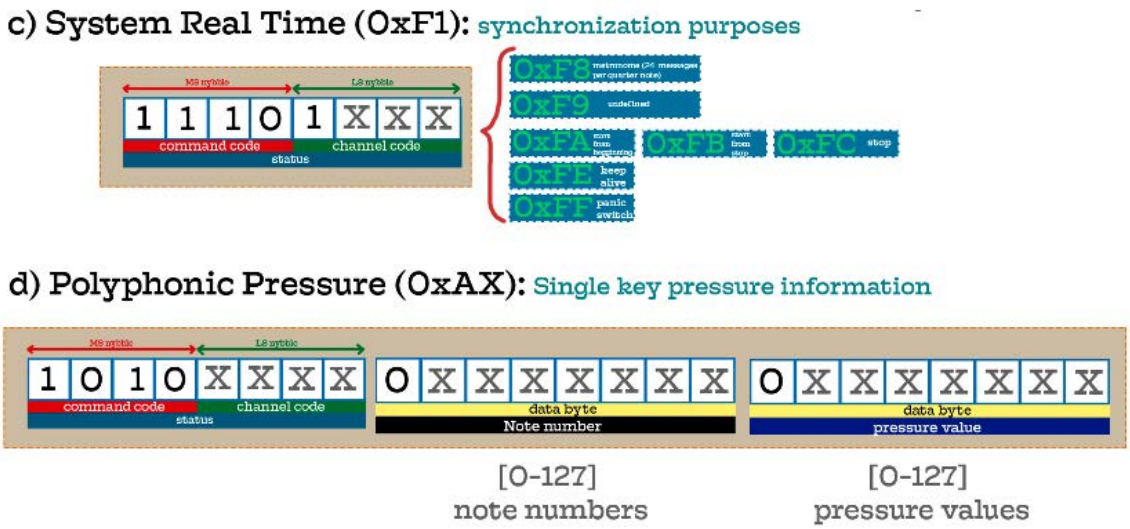


Figure 2.10: *System RealTime* and *Polyphonic Pressure* message schemes

mand code **0xD** and in contrast with polyphonic pressure, channel pressure messages comprise only one data byte, containing the channel pressure value.

- **Pitch Bend**: A spring based mechanism is usually added to a MIDI controller to enable glissando or portamento, so that the notes that are on can be pitch shifted, and then, go back to their original pitch. Pitch bend messages are recognised by the status code **0xE**, and they need two additional data bytes, both of which contain information regarding the bending. there is a total of 14 bits of pitch bending range, taking as starting point 8192 from a max value of 2^{14} and 0 as the minimum value (corresponding to the maximum bend allowed upwards and downwards respectively).

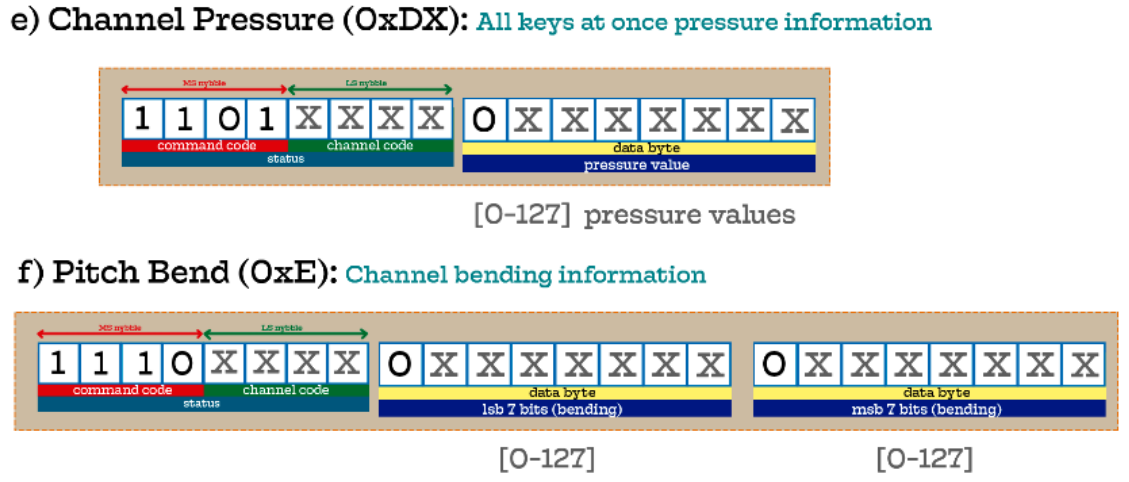


Figure 2.11: *Channel Pressure* and *Pitch Bend* message schemes

- **Control change (CC)**: More parameters regarding sound and expressiveness can be provided through what are called continuous controllers, which are often implemented as a set of additional knobs or sliders. These parts can usually be mapped to control different things, even though there is a standard

specification for them, but it is not universal so many devices allow you to configure them to taste. These messages are recognised by a command code of **0xB**, and expect two additional data bytes. The first data message contains the controller number; that is, whatever is to be controlled, whereas the second data byte is devoted to specify a value to set to the given controlled expressive parameter.

- **Program change**: As there are many synthesizers allowing for different sounds within a specific MIDI channel, we can choose the sound that will be played by stating so using a program change message. You change the patch set on virtual synthesizer to the desired one. A program change message is identified by the command code **0xC** and requires a single data byte, containing the patch number.

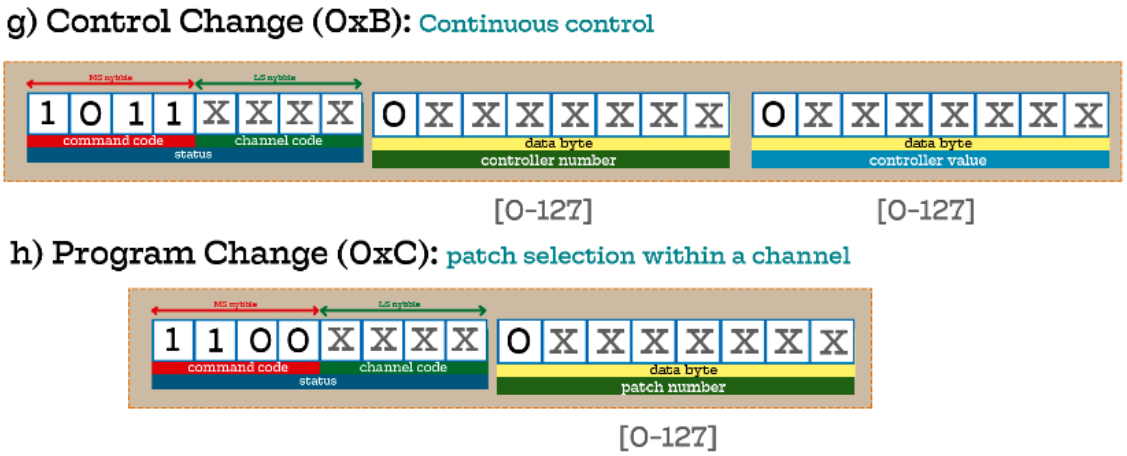


Figure 2.12: *Control Change* and *Program Change* message schemes

- **System messages**: System messages act as an auxiliary envelope for events that don not fit within other statuses. To state that a MIDI message is of this kind, we use the command code **0xF** and we set the least significant nybble of the status byte to the purpose of the message (encoded). As we saw, in case the first bit of the second nybble is set, we have a system real time message, otherwise we may have:
  - Time Code Quarter frame (**oxF1**): Expects one data byte and it is used to transmit absolute time information, in order to enable synchronisation with video playback systems.
  - Song position (**0xF2**): It is possible to construct messages that allow moving around within a song being recorded in a sequencer. This message expects two data bytes which conform an offset expressed as the number of sixteenth notes elapsed from the start of the song.
  - Song Select (**0xF3**): It is possible to tell a sequencer to select a new song from MIDI messages. These messages require an extra data byte encoding the number of the song.
  - Tune request (**0xF6**): (dated) used in analog synthesizers to trigger automatic tuning functionality.

- **System Exclusive**: There are two command codes left to assign according to the previous discussion; those are **0xF0** and **0xF7**. These are reserved to SysEx messages, which can be used to send any kind of data we aim trough MIDI; they can be used for custom purposes, and they can take any length, and follow a very specific structure discussed next:

  There is an opening byte, called Start Of Exclusive (SOX), which carries the 0xF0 as data; then information regarding the vendor is expected, which can be encoded in one or 3 bytes; depending on the nature of the vendor. If the vendor was involved in the MIDI starting era, then, it will suffice with 1 byte to identify it. If the first byte received is zero, then the following two bytes will encode the vendor. Non-commercial entities have a specific one-byte identification number, which is 0x7D.
  After the vendor ID, we can append any data we want as a stream of bytes, which is closed by a new type of byte, End of Exclusive (EOX), characterised by a the value OxF7. We can use vendor IDs to tell whether the data is real time or not, so that we can prioritise accordingly.

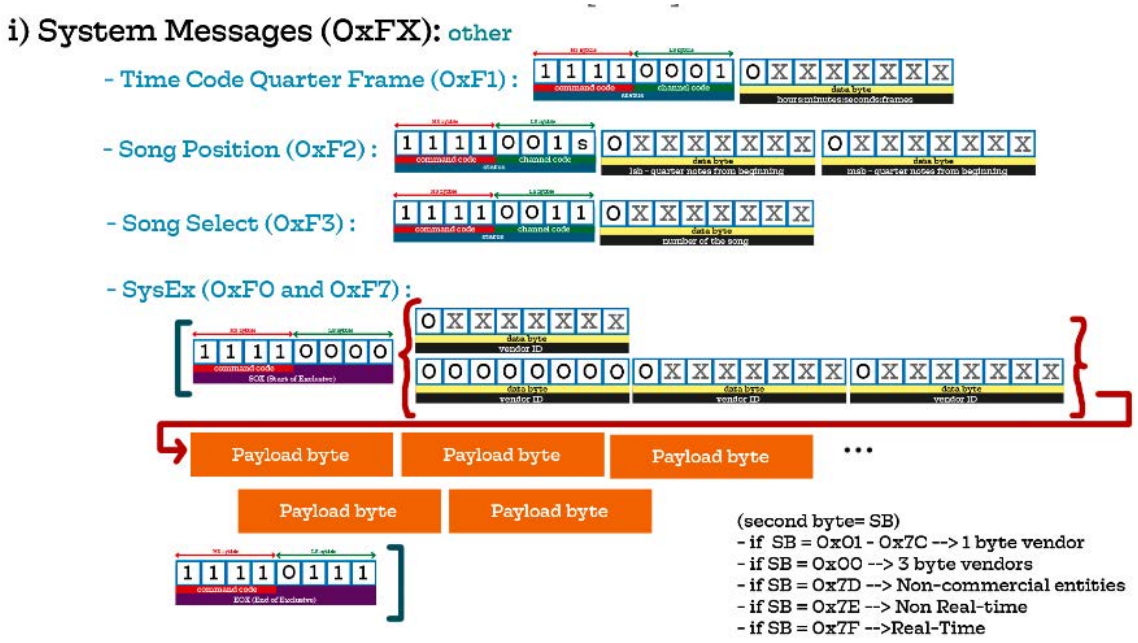


Figure 2.13: *System Messages variations*

2.2.3 MIDI concerns regarding the project

There are many possible topologies for a MIDI system. We are interested in recording MIDI events in a computer sequencer, at least regarding the physical interface that we want to create for this dissertation. Therefore, note that the topology towards which we will design the system later on is known as **Computer-Sequencer**. The computer may include any set of Daisy Chained MIDI output devices (those generating the sounds); or these can be external, as you see in Figure 2.14.

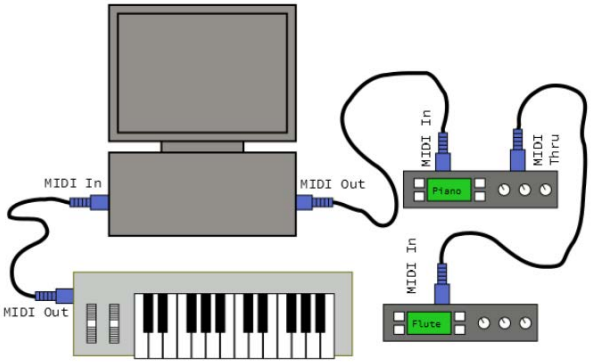


Figure 2.14: Computer Sequencer topology; source [15]

On another matter, **MIDI channels** can be reasoned as a limited number of 16 independent pieces that can be set and produce sounds independently from one or many different MIDI controllers. Therefore, it is not a good idea to think about MIDI channels as independent instruments controlled via MIDI messages from differently shaped MIDI controllers, but instead, you can reason about them as different parts of a music sheet, which are built as different parts of a song but instead of being played strictly by specific instruments of the orchestra, you can later play with the assignment of those, that is to say you can choose which instruments do each part. A channel is an isolated part of your composition, which may be controlled using only status bytes referred to that channel, keeping the rest of the channels unaffected by commands not headed to the channel itself. Channels are slots, different destinations of commands.
Figure 2.15 shows the main parameters that the concept of a MIDI channel handles, each channel can be set to a "program", usually known as "patch", which associates with the kind of synthesizer that will process MIDI messages. The Program Change message allows to change such setting in the synthesizer part of the digital instrument chain.
For each channel, we have the same amount of notes which can be played back on the synth (their state [on or off] is to be monitored by the synth), each logical key has also an associated velocity value which refers to the strength of the press or hit, depending on the type of MIDI controller used.
Regarding the project directly, according to General MIDI, drums are supposed to be mapped to **Channel 10**, even tough many VSTs do not really restrict that much the Channels according to the standard. This consideration will be evaluated later, in the Design phase, to decide whether compliance shall be guaranteed or not.
This whole set of data structures and abstraction layers may be handled later and implemented if necessary on the software side in order to support MIDI input, so it is of chief importance to understand their insights. Extra configuration for sound characteristics can be controlled via CCs (Continuous Controllers), which are usually knobs or sliders that trigger Control Change messages. Examples of these are volume or panning of the sound. You can check all the Standard Continuous controller mappings at [17]. This may be of special interest for enabling configuration of the sound from the hardware interface in the sound-generator subsystem or in order to implement a realistic hi-hat control.

Figure 2.15: MIDI Channels scheme

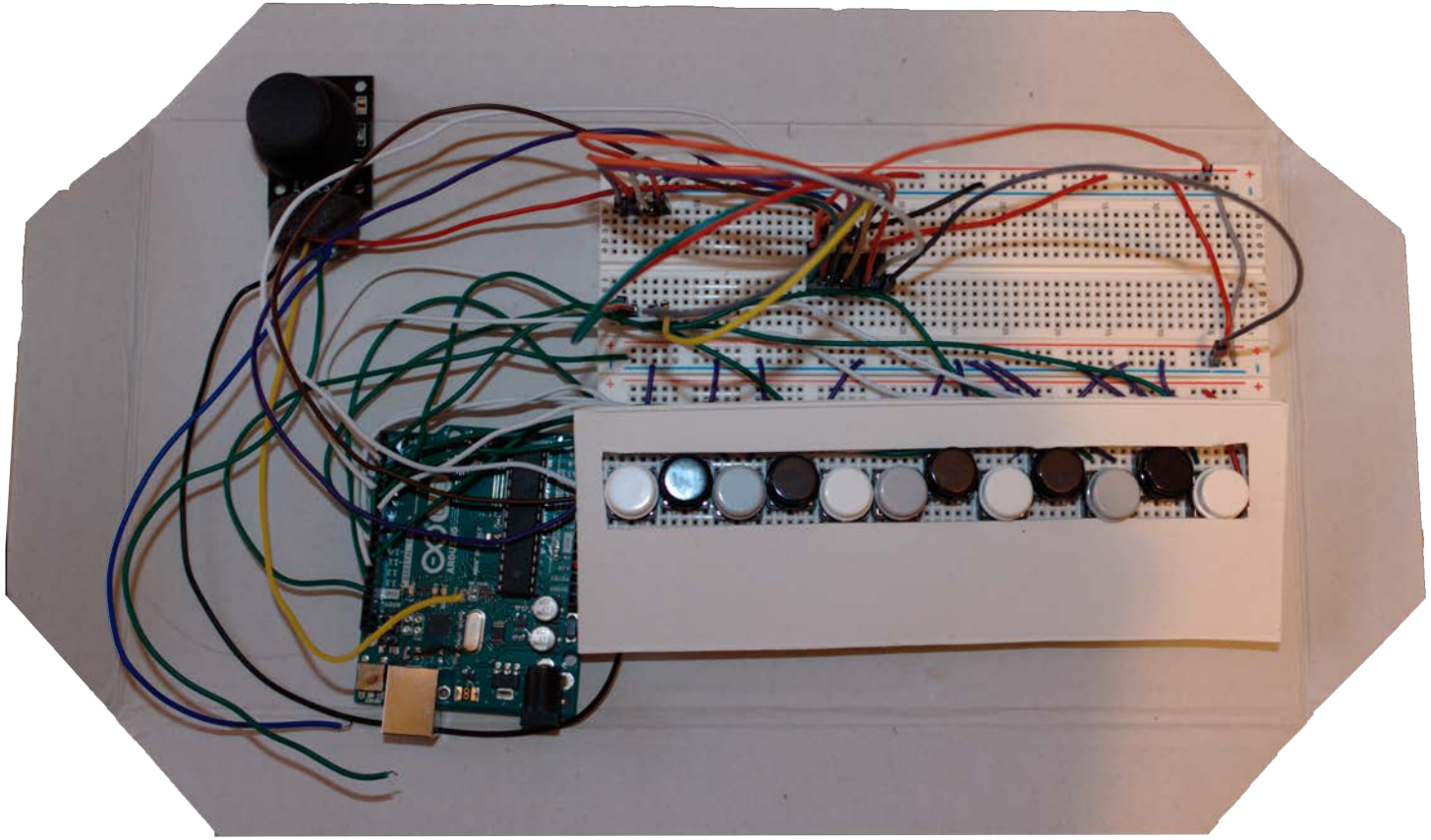


Figure 2.16: *MicroKeyboard* project developed to settle down MIDI concepts pragmatically.

Figure 2.17: C Maj 7 (4,11) chord on a Novation MIDI Keyboard

As a proof of concept, the author of this paper developed the one-octave piano MIDI controller you see in Figure 2.16. So that a basic hands-on exploration was performed to fully understand messages and their purpose within the scope of music composition, spotting those types of messages which the author could be interested in for its use in the final product.
It implemented all the type of messages described previously and used a joystick to provide with Pitch Bend Support among other functionality. The code for the project can be checked at [18].

2.2.4 Main MIDI Limitations and the future of MIDI

This part of the document summarises the main limitations associated to MIDI 1.0, which is the most widespread MIDI version as of this dissertation.

- **Low bandwidth and inherent delay**: Let's just consider we aim to play a total of 10 simultaneous notes at once, testing the isolated delay of bare MIDI, at its baud rate of 3,1250 bauds. This case scenario could be verbosely firmed up as a Major 7 chord with an added tension played in two different octaves; which doesn't constitute a crazy amount of notes to be played at once and involves both hands and all five fingers in the playing process. This example is represented in Figure 2.17 .

  If we consider both no running status and running status for MIDI transmission, solely, *NoteOn* messages and the corresponding delays, we would have the following numbers:

10 notes with no running status involve a total amount of 30 bits to be transmitted per *NoteOn* message. In other words, we would have:

$$10\text{notes} \times 3\text{data bytes} \times 10\text{bits/byte} = 300\text{bits}$$

Those 300 bits require $300\text{bits} * 0.032\text{ms/bit} = 9.6\text{ms}$ to be transmitted sequentially.
Since sequentially means precisely one after the other, real time is literally impossible to guarantee due to non-parallelism. For this scenario, each note will be delayed a total of **0.96 ms** with respect to the previously received one.

Running status mitigates a bit the delay associated to the lack of parallelism but the improvements are not that outstanding. This bandwidth optimisation mechanism reduces the amount of data to be sent from the previous 300 bits to 210 bits, since only a status byte needs to be sent, giving birth to the status and continuous stream of data bits. A complete noteOn message goes first and then, 18 data bytes are sent.
If we do the math:

$$1\text{whole message} \times 3\text{bytes} \times 10\text{bits/byte} + 9\text{Data Byte messages} \times 20\text{bits/DBm} = 210\text{bits}(2$$

This amount of data can be transmitted in:

$$210\text{bits} * 0.032\text{ms/bit} = 6.72\text{ms}$$

Comparing these two approaches, we obtain an improvement of around

$$\%\text{Decrease} = \frac{T_0 - T_{improv}}{T_0} \times 100 = \frac{9.5\text{ms} - 6.729.5\text{ms}}{9.5\text{ms}} \times 100 = 29.26\% \tag{2.4}$$

According to several articles, the amount of delay that most musicians are sensitive to is around 10-12 milliseconds, although it varies according to the kind of instrument and is often associated to the attack time that it has. Singers are the most sensitive to delays regarding buffering, processing and AD converters (as they hear themselves internally before listening to the amplified version of their vocals) yet since we are not interested in Vocals but in MIDI, that discussion will end now, for more info regarding this issue consider going through, under the "Acceptable Latency Values" section [19].

- **No time-stamping, handshakes or error correcting mechanisms whatsoever**: the protocol fully trusts the communication channel, since there is no mechanism used to check for bit errors or correct timing / duplicate messages being received.

- **Problems with continuous controllers**: since Control Change messages don't have any kind of optimisation measure applied to them such as running status, we face a problem here regarding the huge amount of messages that are generated by rotating a knob on a midi controller, since a message (30 bytes are sent per modified value, which increases the delay due to sequential messaging and potentially delays other MIDI messages such as NoteOn or NoteOff. This amount can be reduced by means of programming in the MIDI controller end, minimising the number of messages of this kind sent over the channel per second, finding a balance between precision and usability.

2.2.4.1 MIDI 2.0.

At the time of writing this dissertation, a new version of MIDI was announced to be in a prototyping phase. This new version is mainly aimed at improving the bandwidth problems of MIDI 1.0, which did not allow for real-time control over sound synthesizers, and it also wishes to adapt better to the new technical environment surrounding MIDI [20] . However, no measures were taken to adapt to this new standard, since the project was at an advanced stage when this advances were made public at NAMM 2019 [2].

2.3 Virtual Reality and Augmented Reality Musical Instruments

Back in 1965, Ivan Sutherland conceived an "ultimate display as one that could provide an immersion comparable to that of the real world; where objects could be as usable as real world objects and the computer could control the behaviour of matter [21]. Virtual and Augmented Reality efforts have narrowed the gap ever since, yet there is a long path to cover still.

In recent years, a rapid development of VR and AR displays and technologies; specially HMDs (Head mounted displays) such as the Oculus Rift an, HTC's Vive (Virtual Reality alternatives) or the see-trough display offered by Microsoft Hololens; has brought interest back to the paradigm, since more and more competitive prices for these devices are coming out.

Anyways, new interfaces for sound and music generation have been developed during the last few decades (often referred to as NIMEs), since the augmentation of reality boundaries is specially interesting for creative activities, as it is the case of musical performance and composition.

VMIs (Virtual musical instruments) have become a common tool for producers and songwriters over the world, caging into software diverse models of existing musical instruments and incorporating new features which where impossible to conceive in a real and hardware-constrained instrument (e.g. vibrato and glissando in piano keyboards). These virtual instruments have been around for decades and sometimes generate sounds based on physical models (i.e. they use physics and models of how the sound is actually generated to provide with realistic sounds in real-time); early examples of these VMIs and custom interfaces can be found in Cook's works SPASM [22] and BoSSA [23] or Välimäki, V. and T. Takala project [24](1996)).

Nowadays, VMIs are often controlled by means of MIDI controllers, mostly shaped like a piano keyboard and embedded as VST plugins (which essentially encapsulate a virtual instrument so that it can be used within a DAW), and provide with real-

[2] watch https://www.youtube.com/watch?v=QvJhLQnuktg

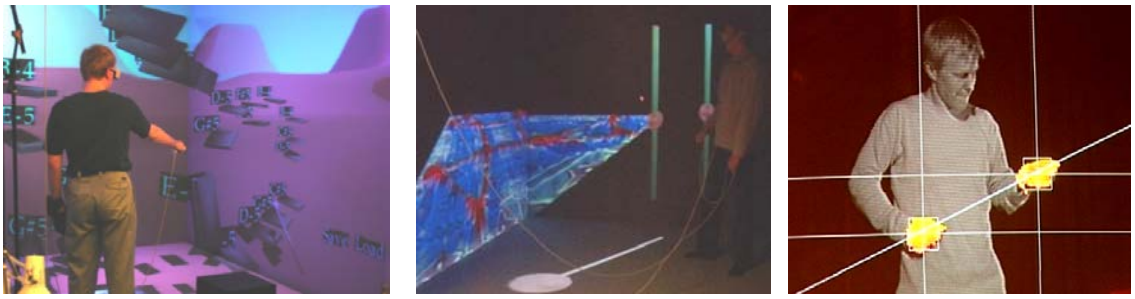
istic sound generation used in musical compositions of all genres.
However, the capabilities of these high-tech virtualized instruments are limited in terms of interaction from the perspective of the user, as interaction occurs and is mapped similarly to how it is mapped in the real world, without leveraging the non-physical constraints characterising virtualization (as the real-world constraints don't apply in the virtual world anymore).
Consequently, and according to Perry Cook's principle "Copying and instrument is dumb, leveraging expert technique is smart" [25] it makes more sense to expand VMIs towards different interaction modes and mappings to the generated sound than trying to simply replicate an existing instrument without modifying the way users interact with it largely. New mappings would allow for a re-contextualisation of gestures in the musical domain and allow for further extension of virtual instruments and the sound they are able to generate in response to user input. This would differentiate new instruments more and more from real-world instruments and eventually, would probably end up being considered instruments on their own (not just improved counterparts) because of the extensions to the real world enabled by these.

To this end, unconventional MIDI controllers, Virtual Reality and Augmented Reality projects have been developed over the years, exploring the potentially unlimited possibilities offered by reality augmentation technologies (VR and AR) in the musical field. Next, a subset of these projects will be discussed shortly, extracting conclusions from different research papers and articles by some of the experts in the NIME field, Computer Music and the AR/VRMIs field (Augmented Reality/Virtual Reality Musical Instruments).

Decades ago (during the mid 2000s), interesting contributions to the VR instruments field were presented in the **ALMA project**, comprising several Virtual Reality Instruments (namely **Virtual Xylophone**, **Virtual Membrane**, **FM Synthesizer** and **Virtual Air Guitar** found in Figure 2.18) that focused on providing with new mappings between gestures and generated output from the instrument that used real-time sound synthesis, contrary to the trend at the time.
The majority of these instruments, as discussed in [26] (2005) did not involve wearing an HMD but instead, a whole room was used for visualisation, where the virtual world was projected and which was perceived as 3D by means of shutter glasses[3]. User input was gathered by means of data gloves, magnetic motion trackers and Computer Vision (this last technology used exclusively on the the Virtual Air Guitar), often used in tandem to detect gestures or direct interaction with virtual objects.

These instruments enabled new interaction modes with respect to their physical counterparts, so they essentially expanded on the possibilities made available by virtualization in clever ways in order to differentiate from their predecessors.
Perhaps the most innovative aspect of these virtual instruments was the ability to control sound generation parameters in real time, as the FM Synth and the Virtual Membrane evidence. In the first project (**FM Synth**) , consisting of an instrument inspired on the Theremin, hand gestures were mapped to different sound param-

[3]shutter glasses: special 3D glasses that work based on LCD technology and trick the user into perceiving 3D from 2D projections via alternating the lens that allows see-through at a given point in time



(a) Virtual Xylophone (b) Virtual Membrane (c) Virtual Air Guitar

Figure 2.18: ALMA project instruments, extracted from [26]

eters (e.g. right-hand-opening was mapped to amplitude of sound; that is to say, volume). In the **Virtual Membrane**[4] project, several parameters could be also altered even once sound had been triggered (e.g. tension, dimensions), allowing for generating sounds that could not be reproduced in a non-virtual counterpart and thus, expanding over real-world constraints.

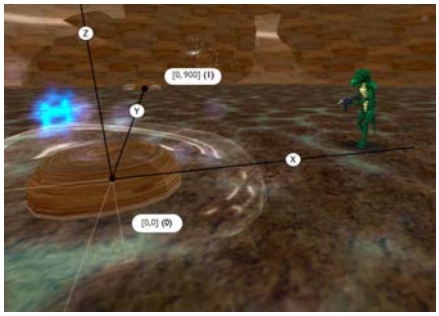
Along these lines, Robert Hamilton would suggest later in [27] (2009) that the lack of physical constraints in the virtual world could be indeed used to re-contextualize the way gestures affect sound within Virtual instruments, even contemplating the possibility of providing the user with sounds which don't match the way we would perceive them in the real world (from a first-person perspective).
An interesting discussion on the possibility of creating a networked shared virtual space is presented as an introduction to the referenced paper. In this interconnected virtual environments, users from different geographic locations would be able to hear the musical events happening in the virtual space in their own physical space, sharing the experience and affecting sound in an correlated way.
Several projects related to the NIME field developed by Robert Hamilton use Q3OSC [28], a special Open-source version of a game engine which allows to generate OSC (Open Source Control[5]) output; that is to say, essentially any varying parameter within a game (e.g. player's position) can be used as an input to the sound creation or shaping process, handled externally by an OSC server that generates the sound according to the in-game situation or events happening at a given point in time.
Among Robert's activity, we find multiple interactive video-games producing some kind of audio output that is somehow interactive, such as ***maps and legends***, a multiplayer shooter videogame which immersed players in a world in which mostly every user action has an impact on the sound generated as output. Within it, paths and sound triggers in the shape of in-game objects are shown to the user to suggest how to use the system in a more deterministic way. The system was devised as a "compositional tool", aimed at being showcased in a specific location, a university

[4]Virtual Membrane: a percussion instrument with configurable membrane characteristics that could be played using virtual mallets

[5]OSC is a protocol for communication among computers, sound synthesizers, and other multimedia devices (primarily targeted at transmitting audio-related data, as it is the case of MIDI) which uses the IP network and UDP packets as communication channel. Unlike MIDI 1.0, its specification stresses on real-time requirements associated to sound generation and control; see [29] for more details

hall, providing a novel musical experience. In that location, audio was spatialized (played back) through a set of speakers that existed in both real and virtual spaces, and which output in the real world was driven by the distance of players to the speakers within the virtual world [30]. As a whole, even though it was not conceived or self-categorised as a musical instrument, this system explores on the NIME research field by offering an interactive and collaborative platform to shape a musical performance from a set of triggerable excerpts to which effects are applied depending on the user actions within the game, making the virtual experience social (which is by the way one of the main problems of Virtual Reality applications in relation to music as discussed in Design Principle 9 for VRMIs in [31]).

Other projects involving Robert Hamilton are **Smule's Ocarina** (an iPhone virtual instrument simulating an actual Ocarina[6] appearing in Ge Wang's article *"Principles of Visual Design for Computer Music"*[7][32] (2014) and ***Tele-harmonium*** [33], an interactive virtual reality project created using **UDKOSC** [8] unlike Q3OSC, Unreal Engine provides much more powerful graphic rendering techniques.



(a) *maps and legends* from [30]

(b) *Tele-Harmonium* from [33]

Figure 2.19: Robert Hamilton's musical videogames

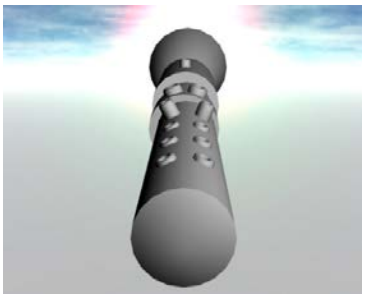
Other remarkable efforts are shown in [34] (2005), where a ***Virtual flute*** with a custom tangible interface is presented to ease interaction and playability of the virtual instrument in a natural way. Similarly to the ALMA project instruments, the Virtual Flute generated sound by means of physical models, emulating real physics but allowing for modification of the size of the instruments to change the output of the model in real-time.In a sense, with this instrument "sound can be created without being limited by e.g. material and form, which makes it possible to play sound, which can not be played on an ordinary physical instrument".

The instrument did not represent the user's hands in virtual space (as can be seen in Figure 2.20a, but a magnetic sensor was used so that the tangible interface and visual representation of the flute could be aligned, adding to the immersiveness of

[6]Ocarina: an ancient wind instrument, it is similar to a flute with an odd shape; see https://en.wikipedia.org/wiki/Ocarina .

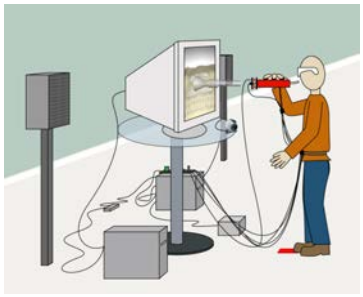
[7]Principles of Visual Design for Computer Music: article including several additional VMIs such as **Magic Piano** or **Magic Fiddle.**

[8]UDKOSC: Unreal Development Kit with bidirectional OSC implementation , a project very similar to Q3OSc[27] which involved modifications to Unreal Engine, a widespread Game Engine



(a) Virtual flute 3D representation

(b) Virtual Flute tangible interface



(c) Virtual Flute set-up

Figure 2.20: The Virtual flute, from [34]

the system. In addition, interaction required actual blowing of the tangible interface, which input was gathered by means of a dynamo turning the spinning motion of a fan into measurable electric current, which made the experience more organic. Perhaps one of the main objectives of virtual flute (and its partner, the ***virtual drum***[9]) was exploring on how visualisation of parameters associated to the generated sound (e.g. colour used as indicator of frequency of the sound wave) could help the performer understand how their actions affect sound easily. Learnability and engagement of instruments are other topics briefly discussed in these research papers, for which visualisation mechanisms enabled by VR could be leveraged.

Some time later, researchers from **Université de Bordeaux** in [35] presented their own implementation of "Reactive Widgets" (firstly described by Levin in [36] [2000]), which were complex 3D objects that provided feedback from specific sound processes and also enabled manipulation of such visualised processes by means of interaction with the reactive widgets. For example, a reactive widget could be a cylinder which varies in shape according to the spectral shape of the sound associated to a given process, and, say, changes its colour to reflect changes in volume too. The whole point of these objects was to provide simultaneous visualisation feedback and control over more than one sound parameter at a time, outstanding from 1D or 2D sliders that control a single parameter (see Figure 2.21). These was a novelty in VR visualisation aspect for sound control, which was yet to be fully leveraged.

Using these approaches to interaction with sound, they created a system called ***Piivert*** (see Figure 2.22), which comprised a head-tracker, a set of pressure sen-

[9]virtual drum: another virtual instrument found in the paper by Gelineck et. al. [34] which user input is gathered by means of computer vision and having similar characteristics to those of the Virtual Membrane in [26]

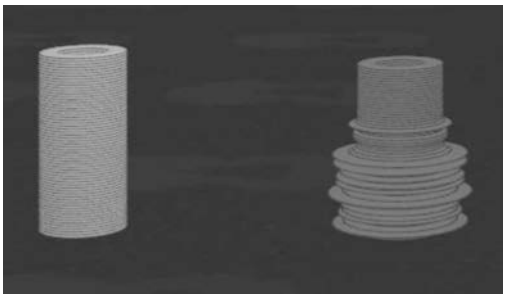


Figure 2.21: Examples of reactive widgets from [37]

sors positioned below the fingers of both hands (a glove-like interface), and passive stereoscopic glasses[10] for visualization of the projected images as three dimensional (thus, no HDMs as we know them were used at this time yet).
Even though the system was not that complex from the point of view of sound generation (since it was limited to triggerable files and effects that could be added to the pre-recorded sounds), we can still consider it a Virtual instrument on its own, providing some innovations for Computer Music in the interaction aspect as well as the visualisation and simultaneous configuration of parameters associated to mapped sound processes.

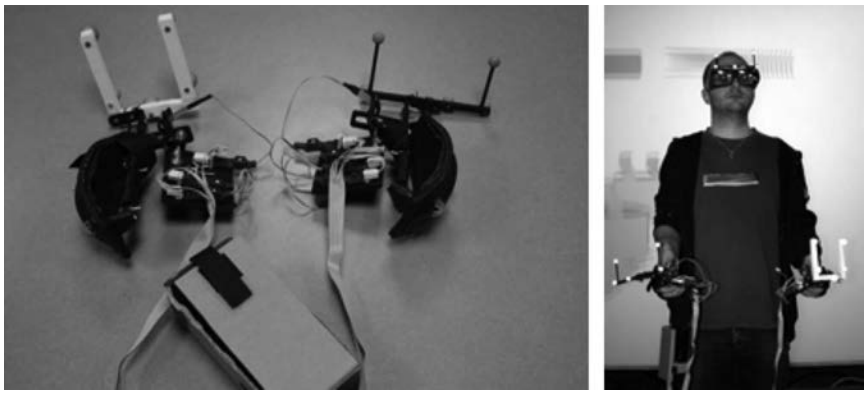


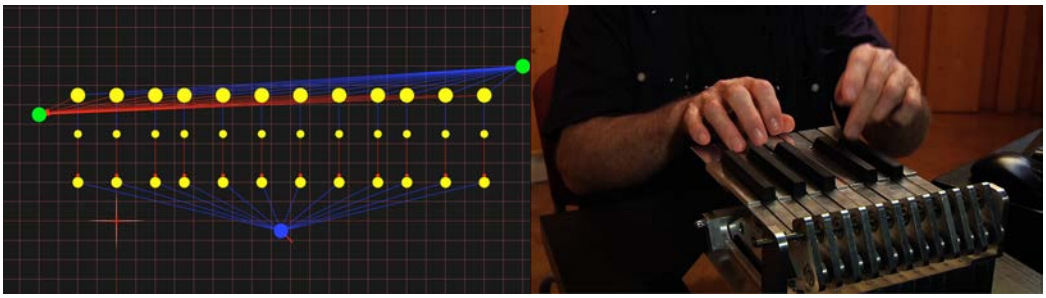
Figure 2.22: Piivert device for gestural interaction from [37]

Perhaps the system described in [38] (2013) is one of the most advanced ones developed in the field of Computer Music (as far as the author is concerned), combining real-time sound synthesis with tangible interfaces that work at high frequencies, all in conjunction with 3D visualisation to provide a believable VR scene that is consistent with the multisensory outputs provided (i.e. tactile, visuals and sound are aligned), and highly focused on the generation of sound; since according to the authors of this paper "systems very rarely consider sound with the same level of importance as the visual or gestural aspects", and they wanted to change that trend and mitigate the effects of latency.

[10]passive stereoscopic glasses: low-cost glasses which have different polarisation on each lens that fits with either odd or even lines of a displayed frame; since half of the displayed information on the screen is meant for the left-eye (odd lines of the frame) and the other half is meant for the right-eye (even lines), glasses allow the user to see slightly different information from the projected overlapping images with each eye

A custom system taking into account different real-time requirements effectively allows a astonishingly realistic interaction from the point of view of the tactile feedback of the device as well as sound generated as output. In addition, the paper does not simply show the creation of an ad-hoc[11] instrument, but instead it shows a complete platform to design and manipulate VMIs without the need for a wide knowledge in Computer Music low-level details, thus, enabling the creation of multiple examples of physical models driven by a high-tech haptic interface (see 2.23b.

However, it is worth noting that the 3D visualisations provided by the generated physically-based instruments are in the shape of dots and lines representing the physical components that would generate such a sound in the real world, that is to say, the represented 3D objects have nothing to do with real instrument shapes, as it can be seen in Figure 2.23a.



(a) *The Piano Model* (b) 12 DOF haptic interface

Figure 2.23: Leonard et al. Piano Model and 12 DOF haptic interface from [38]

More recently, much more sophisticated systems have been developed and discussed by researchers all around the world, many of the projects are present in conference papers, that is, proceedings, such as NIME or ISCCMR [12].

Firstly, directly related to AR percussion is ***V-Drum*** (2015), a CV (Computer Vision) system that allows for generation of sounds by means of air-drumming while located in front of a web camera and involving interaction with both drumsticks and a pedal, whose tips and mallet respectively are coloured differently to ease recognition (see Figure 2.24). This system is actually pretty self-contained, allowing for direct recording of audio and video as well high flexibility and low space demands [39].

Similarly, ***Virtual Drum*** is a system that projects a set of pads over a surface and uses *Microsoft Kinect* [40]. This system is conceived as to explore the ubiquitousness of drum playing and makes use of visual feedback, which is exocentric and could potentially allow several users play concurrently over the projected drum-heads. Thus, Virtual Drums is a proof of concept system that explores in the Augmentation of the reality through projection, and is one of the early systems that utilise image tracking as the interaction source for musical instrument virtualization. As a downside to

[11] ad-hoc system: a system that has been developed towards a very specific objective, and often, implying flexibility limitations

[12] CCMR: International Symposium on Computer Music Multidisciplinary Research

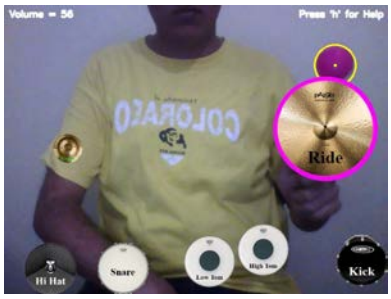


Figure 2.24: V-Drum : An Augmented Reality drumset from [39]

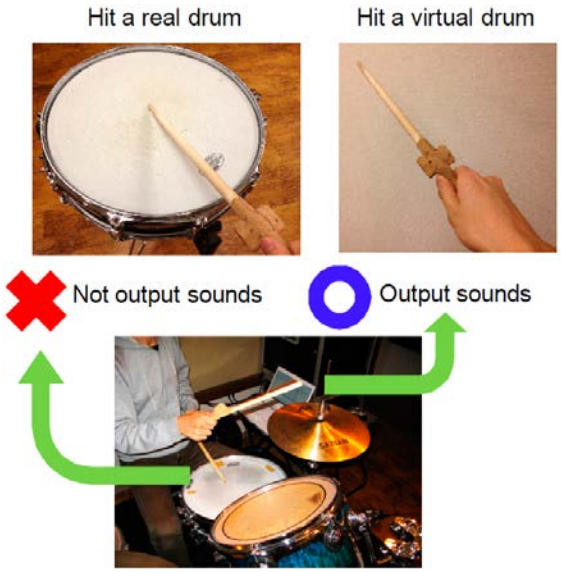
this system, the main drawback is the fact that it relies a lot on future work for a providing the actual portability and ubiquitousness the author claims the system to have, as it uses a projector mobile phones don't currently embed [41].

Figure 2.25: Virtual Drum from [41]

A different approach is executed in ***Airstic Drum*** by Tsutomu Terada [42], which is a system aiming at integrating real drums with augmented virtual drum-kit pieces that ideally could be used interchangeably (i.e. you can either hit your actual, real drum piece; involving no more output than the one coming directly from the drums themselves, or, on the contrary, you can air-drum to obtain an output sound corresponding to the chosen, virtualised, non-physical drumpiece part).
A great point of this system is precisely the enhanced flexibility it offers, allowing for special cymbals to be virtualized in order to avoid transportation issues associated to that, or even better; the system can make unusual and expensive drum pieces and cymbals available for free and ubiquitously. Not only the system is mounted on a pair of drumsticks (which is a great idea for seamless interaction), but it is also wireless and very precise (since thresholds and interaction acceleration curves are adapted to each specific user to fit their technique). Similarly to other projects, it uses gyro and acceleration data to determine the orientation of drumsticks and data gathered from real-drumming interaction and air-drumming are compared to allow their differentiation as accurately as possible. Figure 2.26 depicts the basic idea upon which Airstic Drum is based, i.e. a real drumkit with augmented drum counterparts.

Furthermore, the ***Mixed Reality MIDI keyboard*** is a very interesting project aiming at expanding the capabilities of a conventionally-shaped piano MIDI controller by creating a custom controller which integrates with Unity and provides

(a) Airstic drum tangible interface



(b) Interaction approach

Figure 2.26: Airstic drum from [42]

immersive visualization via the HTC Vive Headset[13] and hand tracking using Leap Motion Hand Tracker[14][43].
This system provides tactile feedback by means of a piano-shaped MIDI controller that exists within the virtual space too, an interface that can grow by means of the flexibility allowed by Virtual Reality (or as the authors of this paper define the system: Mixed Reality or Augmented virtuality in Milgram's terms[21], since the interface is indeed present but as well created in-real time in the Virtual environment). As a difference from previous research papers that used different techniques to gather user input (such as IMUs, data gloves or similar), the Mixed-Reality MIDI keyboard uses a combination of Hand-tracking and recognition of MIDI keys pressed to place the virtual hands properly within Virtual space. It uses the MIDI protocol because of its ubiquity in the computer music interfaces field in contrast to several previously mentioned systems.
Regarding modelling of the virtual system, Blender was used in conjunction with

[13] HTC Vive: https://www.vive.com/us/product/vive-virtual-reality-system/
[14]Leap Motion: https://www.leapmotion.com/

CAD[15], in order to provide with a precise model of the piano MIDI controller that was developed for the project. From the model, both physical and virtual counterparts were created. As one of the last comments within the article, we can find the following sentence "The techniques used for this virtual reality based system can also be applied to Augmented Reality systems such as the Microsoft Hololens to further retain connection to the real world, enabling the technology to be used by musicians in live performances." , which further encouraged the creation of the system covered in this dissertation. Additionally, the code for the project is available at Github[16] and the electronics are based on an Arduino Mega 2560, a platform relatively familiar to the author of this document, so this project was used as a great source of inspiration for this dissertation and as a guide for approaching the project from the very beginning of it.

(a) Mixed Reality Keyboard Virtual Space from

(b) Hardware for Mixed Reality Keyboard

Figure 2.27: Mixed Reality Keyboard from [43]

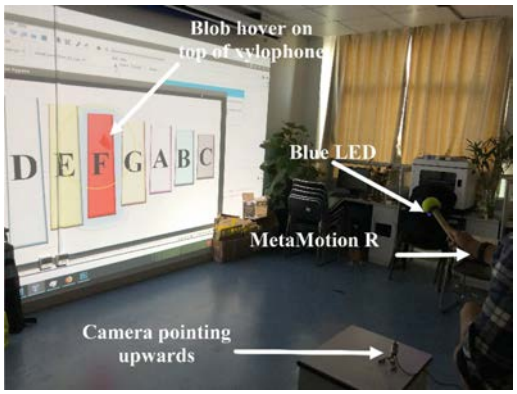
Most recently (2018) I found [44] by Abassin Fangberry and Yanxiang Zang, in which a tangible interface for natural drumming in Virtual reality environments is developed using a wireless IMU[17]to gather data, and CV (Computer vision) to allow interaction with two custom Virtual Instruments; a virtual drum and a virtual xylophone. They study the movement required for a human wrist to execute a hit using a drumstick and model such motion in term of succession of variations in acceleration in a given timespan to correctly detect a "hit" on a drum piece or xylophone plate in the virtual world (see Figure 2.28. One interesting aspect of the developed system was its cost, which was very low compared to that of a real drumkit, and constitutes also an important consideration for this project, since cost of a drumset is high.

In close relation to the system to be developed, but regarding **commercial percussive virtual or augmented instruments** we can outline the following:
One of the first attempts to generate music by means of instrument virtualization including gestural control can be found in the Video-game industry.
Wii Music (2008), a Nintendo game which (very roughly) simulated the experi-

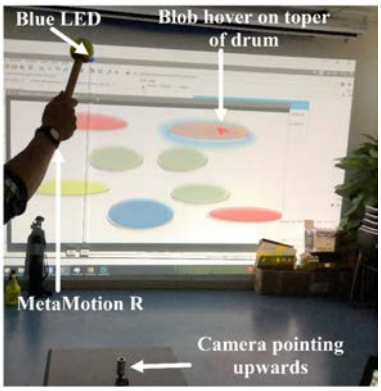
[15]Computer Aided Design; programs of this kind are AutoCAD or ScketchUp programs

[16]Mixed Reality Keyboard: https://github.com/jdesnoyers/Mixed-Reality-MIDI-Keyboard

[17]IMU (Inertial Measurement Unit) a device that allows for measuring acceleration and orientation of the object to which it is attached (by using a triad of acceleromets and a triad of gyroscopes), allowing to determine yaw, pitch and roll (absolute orientation).



(a) UI and setup for Virtual Xylophone



(b) UI and setup for Virtual Drumset

(c) Tangible interface for percussion instruments

Figure 2.28: Virtual Xylophone and Virtual Drumset from [44]

ence of playing different musical instruments (guitar, trumpet, violin, drums, etc) by moving around the hand-held *WiiMote* controller [45]. The game was a nice way of presenting music to young people, but no one would claim that Wii Music was an instrument that could take you further than randomly strumming a real guitar learning or hitting a snare drum completely out of tempo [18].
On the other hand, ***V-beat Air Drum*** [46] was a decently accurate toy device for playing air drums (no longer available), which consisted of a pair of drumsticks and hit sensors for pedals, which allow 8 different sounds which triggered by user action. This device meant a nice exploration step for drum playing virtualization and made evident the possibilities regarding the increased portability that devices of this kind could be offering in the future. Unfortunately, its limitations were too pronounced for replacing an actual musical instrument, it could make practice a little bit more enjoyable though. V-beat Air Drum can be considered as a notable improvement in the right direction concerning its precision and capabilities; this device settled the importance of the interface design with respect to virtual instrument usability, as its hit recognition system was far closer to what it is like interacting with a real drumset than the Wii Music interface was.

Recently, more sophisticated systems that enable accurate drum virtualisation have been developed, such as the popular ***AeroDrums***; which is highly versatile. This

[18]Wii Music review: watch https://www.youtube.com/watch?v=yvHNSTpxjDI

Figure 2.29: V-beat image via https://www.coolthings.com/

system is based again in air-drumming, and provides an interface which is familiar enough to any drummer for straightforward use. A set of special sticks and feet accessories are provided and a camera tracks the movements of the sticks and feet to trigger sound. The tracking is based on light reflections produced by the special gear, which makes them not usable in an actual stage, where lights are changing constantly. It provides a built-in set of sounds, but can be connected to conventional DAWs and VSTs to record drums in your computer easily, so it is a really good solution for practising, playing with your band or recording in a home studio with acceptable quality.
Aerodrums was presented in NAMM 2014 and reviews from hardened drummers were really good, since the system combined accuracy, an intuitive user interface, decent expressiveness and customisation of sounds [47]. In fact, they recently announced they would be extending the capabilities of the system by supporting VR projection of the drum-set. The system is also affordable (185 euros) [48], which is a really good point regarding the importance of this factor to average drummer.

Figure 2.30: Aerodrums tangible interface; image via https://www.amazon.com/

Another Mixed Reality musical instruments worth mentioning is ***The Music Room*(2017)**. ***The Music Room***[19] is a collection of instruments that you play in Virtual Reality. it acts as an accurate MIDI controller that enable strummming, sliding, bending and

[19]http://www.musicroomvr.com/

drummming in a relatively natural way. It looks gorgeous, as it immerses the user virtual space where you can find a set of carefully 3D scanned drums and cymbals from many top manufacturers and play with the aid of the HTC Vive headset and [49]. This system seems to perform good in terms of expressiveness, but the immersion, the air drumming approach, the need for accessories to play pedals and the use of conventional joysticks for playing makes this VR system less familiar to drummers coming from an acoustic or realistically-shaped electronic drumset.

Figure 2.31: The music view in-app image; image via http://www.musicroomvr.com/

2.4 State of the Art Conclusions

This subsection aims to extract a set of chief conclusions from the aforementioned information to give the reader a panoramic perspective about how it all relates and can be applied to the project which is subject of this document.

It is a fact that virtual and physical instruments can be easily differentiated because of the distinctive architectures characterising the two.
Whereas acoustic instruments comprise two components (excitation source, e.g. plectrum; and resonating system, e.g. air, strings, performer's body...) which are heterogeneous and hard to conceptually separate, digital instruments (VMIs) can be split into a gesture controller and a sound-generator subsystem, which are interfaced to each other and essentially can be isolated. This shift on design of instruments was introduced by MIDI and advances in technology and provided new possibilities regarding mappings between gesture and timbre. This is referred to as "splitting the chain" by Jordà in [50], and sets out a advantages and problems:
On the one hand, splitting the chain is an advantage from the point of view of the rapid development of both interfaced subsystems in an isolated way. This means more research is focusing in a single edge of the conceptual whole, so better features and synthesis algorithms have been developed over the years and different interfaces for percussive and non-percussive instruments have been created. The problem with the approach is precisely not taking into consideration design of both ends in tan-

dem. Which would allow for a better matching and usability of the system, taking advantage of as many synthesizer features as possible. This is a relatively unexplored approach to design, that ought to be performed in order to provide an actual complete instrument that goes far beyond the use of a general purpose controller. This is something several state of the art instruments referenced in this document have aimed [34] [26] [38].

According to the diverse projects discussed earlier, we could distinguish two major types of VMIs, according the type of sound-generator subsystem they implement: real-time synthesis and sample based instruments.

- **Real time synthesis VMIs**: dynamically generate sound, usually using physical models, simulating real instruments or creating novel timbres. Examples of these are [38], [34] and [26].
- **Sample-based instruments**: some sound generator subsystems are simply in charge of triggering sample sounds with effects applied to those. This is often the case for virtual drumsets, which handle sets of real recorded drum sounds which are then triggered on user action depending on the received user input. Examples of these are [39] or [37].

This project will not be focusing on sound synthesis but instead it will search for building an integrated instrument that is sample-based, since trying to learn advanced audio synthesis concepts would be too much effort for the sole purpose of this dissertation.

Regarding communication between Gesture controller and Sound generator subsystems, we have found two protocols as standard to transmit audio related data, each one with specific purposes. The use of one of these will be required for design of the solution, but let me shortly summarise the differences between the two:

- **MIDI (Musical instrument digital interface)**: It is a protocol which defines a hardware interface for data transmission, and is based on Serial communication. The data transmitted consists of messages with a limited number of binary fields, processed as integers indicating information about e.g. which note to play, volume, channel, etc. Extensions have allowed it to work over Wifi and Bluetooth, enabling wireless communication.
  It is the standard protocol used by most companies in their interface products. A new version of MIDI was announced and is said to overcome most problems regarding inherent latency, time-stamping and note limitations among other problems discussed in Section **??**.
- **OSC (Open Sound Protocol**: is a protocol created by Berkeley University as replacement for MIDI 1.0, providing much more advanced features in terms of performance, flexibility and networking. The protocol uses the network and UDP packets as transmission channel and is implemented in famous projects

as **ChucK**[20], **MAX/MSP**[21] or **Reaper**[22]. However, it lacks of support from the commercial point of view.

The main question regarding this project is, as a matter of fact, the following:**What is the purpose of drum virtualization or augmentation if there are physical counterparts that do sound realistic?**
Well, there are clear distinctions among drumsets of progressive virtualization degrees. We might distinguish three main types of drumsets in this sense: Acoustic drums, Electronic Drums and AR/VR drums. Next, the basic differences among these three types of instruments from the perspective of the author's own research and state of the art's discussions will be depicted; they are summarised too in Table 2.1.

- **Acoustic drums**: these are the most expensive (when comparing features of all of the competitors), the least portable and the ones providing the highest sound fidelity (obviously). Flexibility is often a problem as adding new pieces to a drumkit involves buying new hardware, set-up and spending a large amount of money (specially in the case of cymbals). Nothing feels as real as hitting a real drumset so the physicality involved here cannot be better, and the drummers on acoustic sets can, indeed, "feel the beat". Performance is great due to the rebound offered by drum pieces themselves which allow for faster hits, and expressiveness is fairly large (dynamics are up to the user) and sound is generated in real time with no delay introduced by virtualisation.

- **Electronic Drums**: these are similar to acoustic in shape and in average are a little bit cheaper. They also provide higher portability with respect to acoustic counterparts, since they tend to be smaller. Sound fidelity depends on the sound generator subsystem that can be implemented in HW or SW, so often these are used as MIDI controllers for VSTs. They provide natural tactile feedback and gestures to play these are fundamentally identical to those corresponding to Acoustic Drums, so precision is also similar, but often introducing extra delays due to buffering on a external computer running the sound-generation subsystem. In terms of technique reuse, electronic drums allow for full leveraging of the performer's drumming skills.
  These kits are much better in terms of flexibility, since it is much easier to remap pads to sounds via software rather than physically rearranging or adding acoustic parts to the set. In fact, these drums allow use of all kinds of sounds, so the drummer is not restricted to non-pitched percussion sounds. Access to one of these can provide with great expressiveness if you get a high-end electronic drum-set (i.e. an expensive one), such as [51].

- **AR/VR drums**: discussion on these drums is vague due to the multiple solutions provided, each one with its own pros and cons. On the one hand,

[20]ChucK: a sound-synthesis-oriented programming language developed by Ge Wang and Perry Cook; see http://chuck.cs.princeton.edu/doc/

[21]MAX/MSP: a visual programming application that allows complex sound generation by means of linking boxes which are in reality independent modules; see https://cycling74.com/

[22]Reaper: a DAW just like Cubase, allows for MIDI recording, editing, processing and mastering sound

price varies a lot depending on the type of Hardware involved in the system. If they use the latest immersion technology prices are high but options like [44] show that affordable instruments can be created. Either way, prices on devices like Hololens, HTC Vive and similar are bound to lower as technology progresses, so all of the AR/VR instruments based on HMDs will eventually become relatively inexpensive. Most solutions shown in the provided State of the art panoramic are cheap if we consider the flexibility those provide as well as the huge portability enabled by these.
Tactile feedback is a lack in most instruments to date though, due to the interaction mode they use: air-drumming. This approach makes skill reuse only partial because experienced drummers are required to change the way they perform and accustom to be the ones stopping the virtual drumsticks once a hit has been detected, which is known to influence their performance (at least without extra practice to settle down a new interaction technique).
Flexibility is large and similar to that of electronic drums, but enabling novel mappings of whole-body gestures to sound parameters, which would expand the possibilities of a real or electronic drumset greatly.

Type	Acoustic Drumset	Electronic Drumset	AR/VR drums
Price	High	Miedum-high	Widely ranging
Flexibility[23]	Low	High	High
Expansion Cost	High	Medium	Low
Portability	Low	Medium	High
Expressiveness	High	High	Potentially the highest
Whole-body gestural control	No	No	Yes
Physicality	Natural	Quite Natural	Often Low
Performance precision	High	High	Medium
Interaction Mode	Real-HW hitting	Real-HW hitting	Mostly air-drumming
Skill reuse	Complete	Complete	Partial[24]

Table 2.1: Comparison among types of drumsets

As a whole, research on previously developed instruments and design guidelines has allowed the author of this document to isolate a set of VMI key considerations to be taken into account in the development of this project.

VMI Key design considerations:

- **Low latency is a must**: physical instruments introduce no delay due to buffering or virtualisation, so the the time between interaction and sound shall be minimised on a VMI, ideally to values below 12 ms for most instruments

(which is often even lower for percussion, when latency is often noticeable at around a 6 ms delay) [52].

- **High Expressiveness**: dynamics in music performance and composition are very important, so a VMI shall allow for a high level of expressiveness, ideally similar to that of a real instrument or exceeding its expressiveness capabilities.

- **Expansion of real-world counterpart capabilities**: real-world instruments are probably already the best they can be, and replicating them in the virtual world via physical modelling would not take advantage of the possibilities offered by digital environments. As Perry Cook suggests in [53] , new algorithms may suggest new control metaphors and new hardware might as well suggest new synthesis aspects; so it may be a good idea to design both controllers and synthesizers in tandem, avoiding simulation of real world instruments and creating an instrument that can be considered as a new one on its own right.

- **High portability**: specially for drumkits or percussion as a whole, portability has always been an issue, since the dimensions of drumsets are large if we compare them to those of any other hand-held instrument. Virtualisation of these instruments would allow for an enhanced flexibility regarding the number of sounds that a drummer can perform along a performance and would minimise the transportation problems and limitations involved in touring, when you are required to carry your high space-demanding instrument with you to every gig.

- **Skill reuse**: A VMI shall be easy to learn, at least as easy as it is to learn a rea-world instruments. In case of percussion, it would be natural to allow for technique reuse, so that the performer does not need to learn how to play a virtual counterpart to produce sounds similar to the ones he can already generate using Electronic drumsets, for example.

- **Tactile feedback**: According to multiple experiments [54] and among others [55]; there is a need from performers of physicality, that is, having something they can physically interact with; be it a keyboard, a stick or some kind of non-virtual, tangible hardware for sucessful and accurate interaction. This is partially, in my opinion, due to the concept of affordances[25] of an object, whose are biased from interaction with the real world (i.e. a stick suggests that you use it to hit something whereas the air has not often got that associated use).

- **Minimisation of cybersickness**: users trying out VRMIs and other Virtual reality products have complained about adverse effects on their bodies after use of HMDs. These effects include disorientation, headaches, sweating, eye strain and nausea and have been studied to determines their cause; being the most popular explanation to it a conflict between the visual and the vestibular senses [57]. The development of an augmented instrument shall take into

[25] Affordance: a concept introduced by Gibson in [56]; all transactions that are possible between an individual and its environment; e.g. a chair suggests you to sit down on it

account these potential discomforts and try to minimise them by reducing multisensory conflicts to the extent possible.

- **Gestural mapping**: gestures make a great impact in the sound of a real instrument and are crucial for expresiveness and pitch stability (as in the case of a flute; where both the blowing strength and the hands position and pressure make the sound largely different). Thus, its is important to leverage gestures and expand on how these afffect sound within the virtual space on VR/AR instruments.

- **User technique tailoring**: playing techniques vary extensively among performers, specially regarding non-pitched percussion (drums); which is evidenced by the multiple "correct" techniques for holding the drumsticks (e.g. american grip, german grip, traditional grip ...[58]), so a VMI shall be configured so that the sensitivity of sensors involved or associated software adapts closely to the technique of the performer. An example of a project considering this fact is [42].

- **Engagement**: playing an instrument should be an entertaining experience and VMIs have a great potential for providing visual guides to playing, making the experience of practising, learning and composing much easier and engaging. A project that utilises visualization for teaching and providing cues about accuracy of the learner is **HoloBeats**, an augmented instrument based on markers placed over an electronic drumset [59].

- **High level of Presence**: One of the main purposes of VR from its birth is immersion, usually achieved "by removing as many real world sensations as possible and substituting these by sensations corresponding to the VE". This definition does not make that much sense when dealing with the AR paradigm, which aims to enhance the real world. Either way, having the user's hands present in the virtual environment can help building such sentiment of being "within" the virtual world.
  On the contrary, for AR experiences, the ability of keeping the user attention in the objectives rather than distracted by augmentation is crucial and shall be imposed as a chief application objective.

- **Social interaction**: As VR often involves the use of HMDs, it becomes harder for audiences of VRMI concerts to share the performer's experience. Allowing for joint visualization or co-performance would potentially solve the problem. It could also be solved by means of AR devices running a visualization application showing the audience precisely what the performer is doing.

Chapter 3

Analysis of the problem

This chapter provides a thorough description of the system in three chief levels of abstraction; going from a high-level picture of what the product to be built is about towards a formal definition in the shape of two layers of requirements.
Once an complete introduction to the system has been depicted, the reader will be presented with a User Requirements Specification (concreting the capabilities the user shall be able to perform by means of the system) and finally, System requirements will be covered, which provide a more detailed vision of the system while maintaining a design-neutral approach.
As culmination for this Chapter, the reader will be presented with a traceability Matrix plotting User Requirements against System requirements, to ease tracing errors or noncompliance back to a requirements specification problem.

Section 3.1 goes through a detailed depiction of the system objectives to be achieved, stating how the system has been conceived and showing the constraints imposed over the project development and environment towards which the project is targeted.

Section 3.2 covers the formal specification of User requirements, among which we can find Capability and Constraint Requirements respectively, gathered consciously and reviewed along the project development phase to ensure compliance.

Section 3.3 goes further in the elicitation process and provides a set of baselines for the design and implementation process, that is, the System Requirements specification and additionally generated items that clarify what the system shall do and under which constraints.

3.1 General Description

This section provides an overview of the project and the system to be built, limitings its scope, stating constraints imposed over the development and describing the environment and target users of it.

3.1.1 General capabilities of the system

The centre of this dissertation is the creation of a complete electronic musical instrument comprising both hardware and software that allows triggering drumset sounds in the same way a commercial Virtual Instrument like Addictive Drums 2 [1] does in conjunction with an electronic drumset like say the Roland V-Drums TD-17KVX [60] or any cheaper counterpart.
Thus, like any state-of-the-art electronic musical instrument, ***DrumVR*** , is composed of two main parts, which are ideally independent from each other and which are linked by means of the MIDI protocol. These are often known as the MIDI Controller and the Sound Synthesizer.
Consequently, the system is split into two clear subsystems from the very beginning and this is important to understand the system concept as a whole, conformed by a custom made MIDI Controller and a Desktop program producing sound derived from user's interaction with the MIDI Controller, showing a graphical representation of such interaction.

DrumVR is not just a replica of some other system carried out for the sole purpose of getting to know the insights of similar devices and programs, that is, the goal is not to build a cheap alternative to electronic drumkits, or simply a professional-looking DIY electronic drumset.
Instead, the present system has been developed to provide an innovative approach to MIDI controllers targeted at interaction with non-pitched percussion-oriented virtual instruments, by means of moving away from trying to replicate a real drum with hidden sensors in it and building most or all the interaction detection into the real triggering elements of a drumset: drumsticks and pedals (which are the ones actually driving the interaction in all cases, and the ones whose flexibility is potentially unlimited).
From the perspective of the Synthesizer part of the system (better called sound generating end since it does not necessarily create the sounds from scratch), its main purpose is to allow such a MIDI controller (providing very few real-world cues about the interaction being performed by the user), to present the user with a direct mapping, both visual and auditory, from the interactions they are performing.
Summarising the previous verbose description, you can find a simple scheme that may give some light to understand better the system concept:
On the one hand, the author will develop a Hardware interface belonging to the *NIMEs* (New Interfaces for Musical Expression) that will be used as user main input to the sound generating system, mapped to the second main subsystem to be developed in this project. This interface is to be composed of four main sources of interaction, one per limb, two drumsticks and two pedals.
On the other hand, a sound generating system and graphically-enhanced software program will be developed enabling the generation of different sounds according to different inputs of the user by means of the MIDI Controller system described briefly above. Additional configuration shall be enabled to the user to allow customise how sounds are assigned to specific gestures or interactions with the system.
Figure 3.1 shows a draft aiming to ease the comprehension of the textual descriptions provided herein.

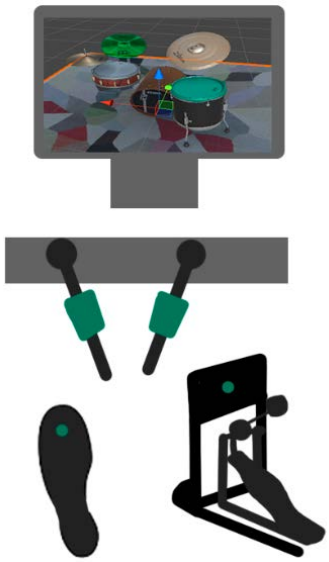


Figure 3.1: Sketch of the devised system

The main capabilities of the system will be enumerated next, showing the processes enabled by ***DrumVR*** :

- Allows users to generate sounds by hitting surfaces that do not include sensors in them.
- Allows users to leverage real drumming skills by providing an interface which resembles that of a non-MIDI counterpart.
- Overcomes portability problems of an acoustic or electronic drumkit by limiting the required hardware pieces for interaction to four.
- Overcomes the flexibility problems of a traditional, fixed-size drumkit, which requires physically expanding the number of drums or sensor-enabled pads in order to have room for more sounds within a drumset.

These capabilities can be leveraged by users in multiple situations, among which we could outstand the following:

- The system can helps users to practice in a non-prepared environment, that is, without requiring a really specific setup to start playing drums that sound like "familiar drum sounds" (i.e. snare drum, hihat, crash...etc).
- The system can help users to get used to play on different drumset configurations by means of distinct custom assignations from interactions to sounds.
- The system can help users develop coordination skills on a very space demanding environment, providing cues about when the user is making a bad move. It is very hard for drummers to practice limb independence without matching sounds to compare movements against.

- The system can help users on a limited budget to access a great practising tool for free, using the open source code referenced in this dissertation.
- The system can help users to understand how MIDI musical interfaces and Virtual Instrument software works and that an ultimately give them background knowledge or inspire them to play around with configuration of such programs and HW devices.

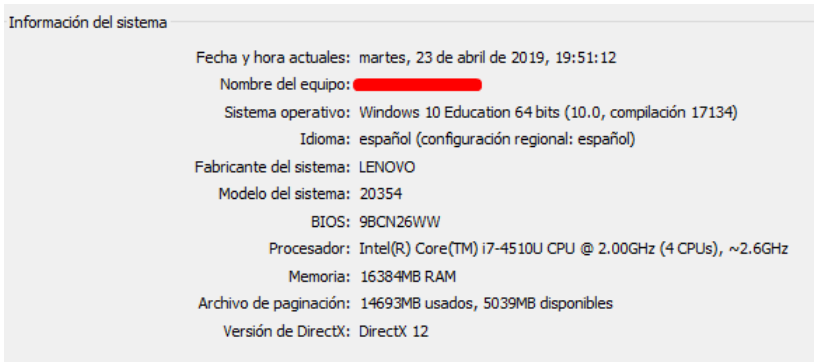
3.1.2 User Characteristics

DrumVR is mainly targeted at drummers or aspiring drummers, so it is assumed that the users are familiar with a real drumset or know how interaction with one of those works. In order to get the most out of the system, concurrent interactions should occur, similarly to what happens when interacting with a conventional drumset. Familiarity with a drumset is specially useful for interaction with the MIDI counterpart of the Hi-hat pedal, which enables an increased executable set of sounds.
Essentially, only very basic coordination and proprioception skills are required for satisfactory interaction with the system. In addition, it is worth mentioning that this system is designed assuming the user is, indeed, capable of interacting with the system using all four limbs. Disabled people may have trouble getting the most out of this system even though it is possible to do a work-around to allow complete interaction for these special cases, and accessibility can be improved in further iterations over the system. In general, most users who are unfamiliar with drums may find the system cumbersome at first, but its simplicity is enough for any novice user to understand how interaction with the system works.
Regarding expectations from users, the system is very likely to be criticised due to the latency introduced between hit and generated sound, which may even be hard to tolerate for certain users, most likely, experienced drummers. However, the focus of this system is not put into providing an optimal performance (since even the MIDI protocol is slow by definition), but demonstrating the feasibility of creating such a system which, once polished could overcome many of the limitations current MIDI interfaces and systems have, discussed previously in Section 2.2. On the other hand, the system aims to be flexible enough to provide experts with customisation and casual users with a plug-and-play experience. This is an important aspect to consider given the wide range of drummers that exist, varying in level, preferences and artistic tastes, who must be satisfied by supplying the musical instrument with a relatively wide set of options.
In this sense, experienced drummers may ask for changes in drumset configurations, addition of drums, inclusion of own sounds or similar queries. In contrast, novice drummers or casual users may simply expect the system to produce a varied set of sounds by default, so that they can play drums straight away.

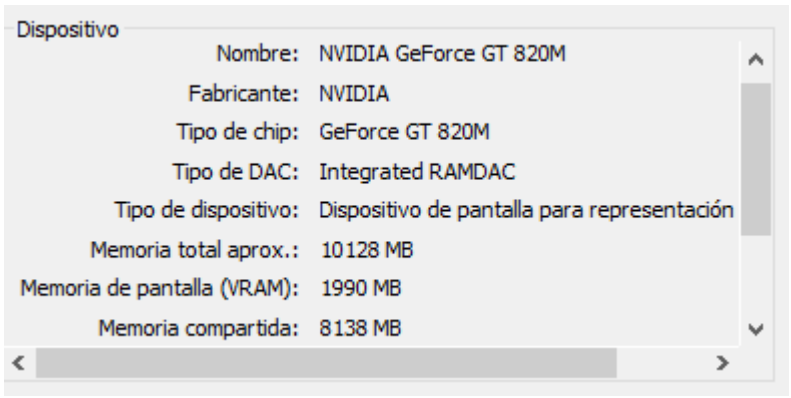
3.1.3 General Constraints

This section covers the main limitations regarding development of the project, covering constraints in relation to equipment, expertise of the developers, time and nature of the project to be described in this thesis.
Regarding available facilities for development, we can list the main building blocks of the author's setup.
The personal computer at hand used to develop the project comprises a **Lenovo Z50-70 laptop**, with the specifications that are shown in in Figure 3.2.
Such computer equipment suffices for development but is likely to slow down the

Figure 3.2: Laptop Specifications



(a) Laptop Specifications



(b) Laptop Graphics Card specs

process due to its age (more than 4 years old), as it is known to have issues running programs of special importance for documentation, such as *Adobe Photoshop* (used for the creations of explanatory diagrams), as well as 3D modelling software (such as *Blender*[1]), Game Engines or Emulators (e.g. *Unity*; *Hololens Emulator*).
Difficult access to University facilities is also a restrictive factor regarding the creation of the project, since due to personal reasons, the author will not live in Madrid neither he will live in the outskirts during the creation of the project. For this reason, rapid prototyping and support for HMDs (which are too expensive for a student to buy for development) or see-through devices like Hololens may be impossible to achieve on the short timespan planned for the completion of this work.

[1]Blender: https://www.blender.org/

On the other hand, this project has the additional complexity of being a **research work**. Thus, since it is not a verbatim replica of an existing system, feasibility estimation becomes harder to perform and therefore, much more time was devoted to solving this kind of issues, slowing the development process as well as elicitation down. In addition to the complexity inherent to research work, the whole process of writing, researching, designing, developing and testing the software described herein is to be done by a single person. This forces the author to show a relatively high versatility in very distinct aspects of the Software Lifecycle; which means that it will be required to spend a lot of time exploring areas to which the author may not be accustomed to.
We may add to this composite situation a **short knowledge-base** regarding the field to be dealt with in this project. This is a manifest characteristic of barely any college undergraduate, since their skills are inevitably limited. Owing to this fact, chances are that the project will become even harder to manage regardless of the initial, perhaps naive, intentions of the author.

Regarding schedule limitations, there are several aspects to cover in this sense.
In the first place, the project had to be developed in spare time, considering the author was working part-time during the creation of the system and its documentation. This imposes a tighter limitation regarding general objectives, which had to be reduced in order to provide a finished whole while guaranteeing quality of the delivered items. At most, three or four hours could be devoted to the project per day, being very optimistic. Nevertheless, this project was planned to be developed in around **10 months** (see Section 6), which is a very limited amount of time, specially due to the short expertise of the author with respect to the technologies being handled during the development process. Such amount of time is clearly a small time-span when you have to make sense of what you want to build, stating whether it is feasible or not to build it and make design decisions you are not sure about all the time, since you are almost oblivious of best techniques or practices regardless of your efforts to get to know as much as you can about the technologies. Since options seemed unlimited at the beginning of the project, one of the problems regarding planning is precisely estimating how much time it would take to choose among alternatives, how much time it would take to learn certain skills and whether, eventually, it would be possible to fulfil the objectives enumerated in the first place.

Lastly, this document was developed in LaTeX , which inherently implied an additional complexity and required booking more time for the documentation phase, since it is necessary to get accustomed to writing and editing documents with this programming language, skills that are expected to be useful for the future along with the knowledge extracted from the creation of the system depicted herein.

3.1.4 Operational environment

The product generated as outcome from this process is targeted at a very specific setting, which is an indoors home studio or a small bedroom within a flat. This is the target environment due to the typical constraints owning a drumset involve, since they are very space demanding instruments. Thus, a typical user of the system would be a musician or aspiring drummer which either does not own a drumset or cannot fit it within its working space. Consequently, the operational environment constraints the design of the system towards having a relatively small size compared with that of a typical drumkit.
The main desirable context of use of the system is practice, since recording is not defined as a primary goal due to its increased complexity. A user may use the system to work in rudiments all over a simulated drumset whose flexibility is much higher than that of a non-virtualized counterpart and whose dimensions will be considerably smaller.
Consequently, this system is not expected to be used in a musical performance, at least in the first set of iterations, since no guarantees are made regarding overall delay, which plays a huge role in performance with other musicians and influences the ease to keep a straight tempo.
The conceived system requires a set of hardware pieces to get the most out of it in terms of seamless interaction. The user is expected to own the following material or something similar to improve ergonomics and similarity towards playing real drums. nevertheless, these are still optional to obtain a satisfactory interaction (even though the way user produces the input becomes inevitably different from that of drumming in an ordinary drumset.
The reader will find several alternatives as well as the products owned by the author of this dissertation for completeness, so that an estimation for the cost of the setup can be later performed.

- **A bass drum pedal**: In order to make interaction as close as possible with respect to a real HW, the best configuration in my opinion utilises a pedal, either double or simple. Prices are varied with regard to pedals but some affordable ones are Pearl's P-530 (40 €[61]) or Tama's HP30 (53 €[62]). Regarding the author's setup, he owns a double pedal, Tama's Iron Cobra Jr. Limited Edition Double Bass Pedal (which is more expensive, around 140 €[63]).

- **A bass drum practice pad**: as support piece for the kick-drum sensors, we will use this piece of hardware, where a sensor will be hidden under the foam. Some alternatives in this section are Millenium's BDP-S Bass Drum Practice Pad (33 €but not suitable for all pedals [64]) or the one the author owns, Evans' RFBass Bass Pedal Practice Pad which costs 88 €and is quite silent [65].

- **A "snare" practice pad**: in order to use it as a physical reference for your playing. The author owns Meinl's MPP-12-JB 12", which is around 35€and very durable [66] , but you can buy the cheap Thomann's Sticky Practice Pad for around 13 €.

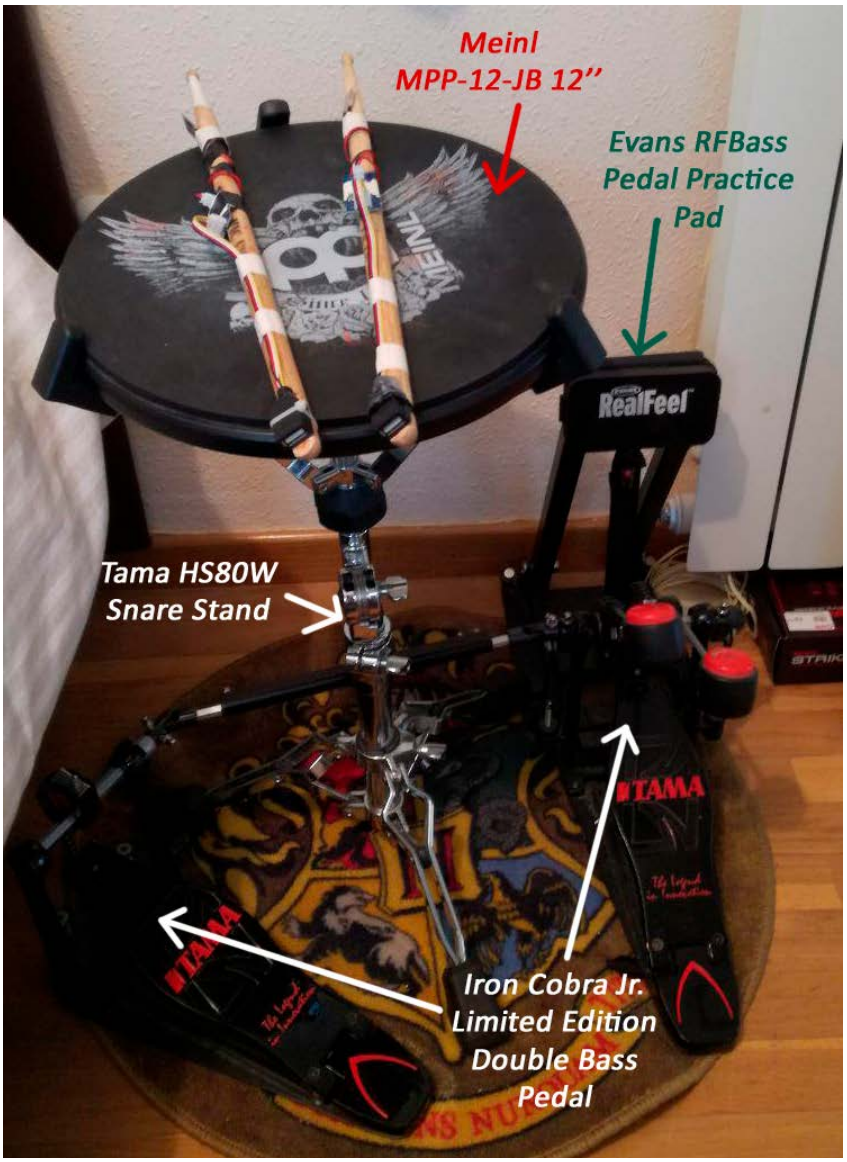


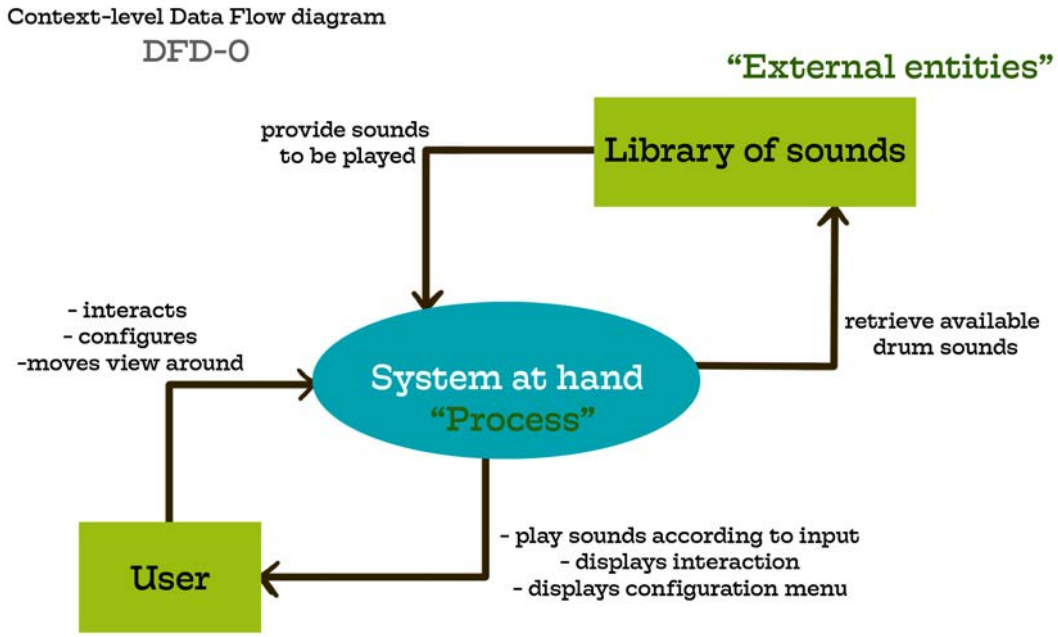
Figure 3.3: Ideal operational set-up

My setup also includes a Tama's HS80W Snare Stand to adjust the height of the practice pad based on my preferences [67].

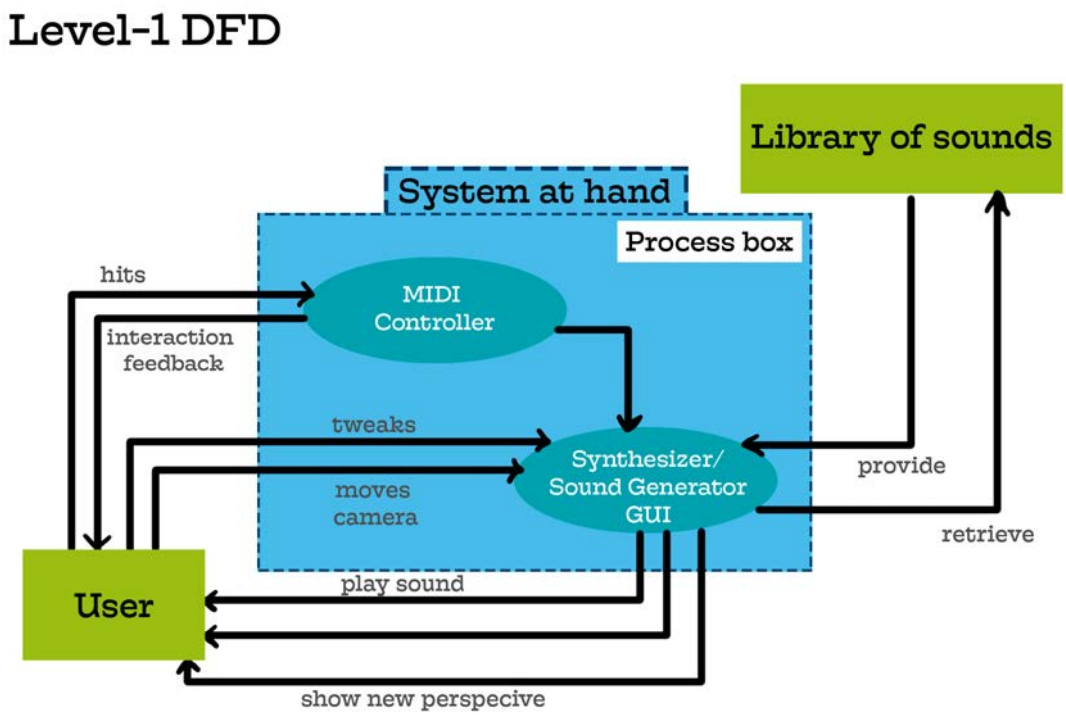
In order to provide a more concise and clear view of the system and its architecture at a high-level of abstraction to any reader, a Context diagram is provided in Figure 3.4. The Context diagram (also known as Context-level Data Flow Diagram) is decomposed into-two levels of abstraction, namely Level-0 and Level 1). The first one is an extreme simplification of the system to be developed, with arrowheads showing the kind of information that is exchanged between different parties involved in achieving a proper execution and the direction of the information flow.

Level-2 diagram goes further and gets more precise about the internals of the system to be developed showing that this system follows the split chain paradigm noted by Jordà [50]. In this case, the oval shapes constitute two separate processes which communicate with each other but are considered part of the system to be developed

so they are located within a "Process Box", the dashed box containing both ovals. A better understanding of the system can be obtained from this second one but the Level-0 diagram was included for completeness.
In addition to the context diagram, a **System Block Diagram** showing basic processing insights is provided, so readers can understand the expected basic behaviour from the system. This part will also be used as input to the requirements definition and design of the system, so it can be considered as an early form of elicitation, keeping the description implementation-neutral. As you can observe from Figure 3.5, a Hardware-based component is spotted as well as a software-base subsystem producing the sounds and most of the interaction feedback.



(a) Level-0 DFD



(b) Level-1 DFD

Figure 3.4: Context Diagrams

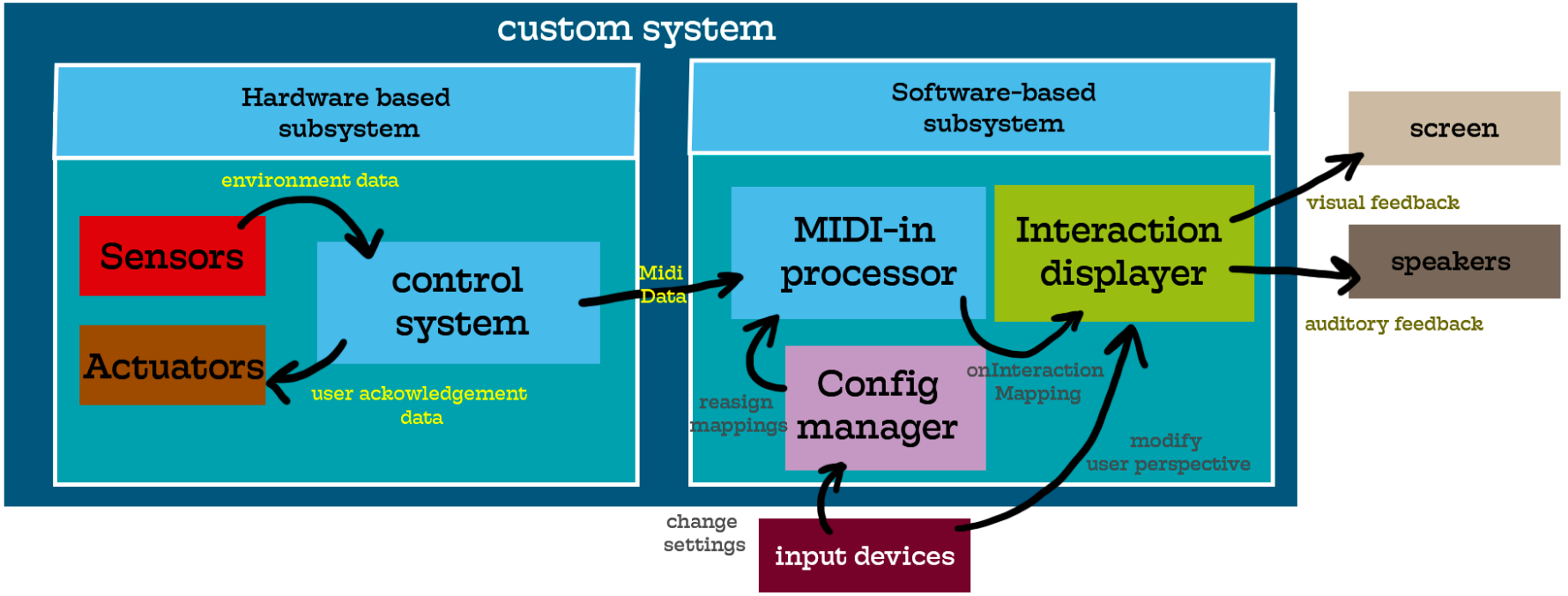


Figure 3.5: Block diagram

3.1.5 Product Perspective

This project is **not** conceived as a **stand-alone** system even though this is an ultimate goal in an advanced iteration of the project. Therefore, middleware can be used to speed up the development process and delegate processes to already created solutions which even though open-source, require a great expertise to be included in the implementation of this system.
The system is not a replacement for an IT system either. However, it aims to substitute in some way an acoustic or electronic drumkit, thus, constraints are inherently imposed as this objective is included in the requirements. Besides trusting on third party-software for interfacing, open source and non-open source software were used along the development process among which we can find Unity, *Arduino* SW libraries, *Arduino Hardware*, *Blender*, *Photoshop*, *Hairless MIDI* and *LoopMIDI*.

3.1.6 Assumptions and dependencies

The present project has been carried out under uncertainty and assuming a set of statements to be true, the following list depicts the main considerations assumed when the project was started:

- A user is expected to have a basic understanding and familiarity with interaction with a computer-based system (i.e. user-Level skills on a Windows operating system).
- The protocols used as well as the hardware involved in the development of the project does not get dated before the dissertation comes to an end.
- The middleware or add-ons used along the development of this project introduce no errors in the project itself and have been already tested before being let available to the public.

3.2 User Requirements

The current section aims to provide with the thorough description of requirements as extracted directly from the target user of the system.
Two types of requirement will be specified herein:

- **Capability requirements**: those that state the actions the user shall be able to perform to achieve an objective. These describe functions and operations that shall be enabled to user. Clarifications on the quantitative verifiability of the requirements will be provided.
- **Constraint requirements**: comprise specifications with regard to how a certain need or objective is to be achieved. This type of requirements states

how software is to be built and operated. Restrictions of this kind may be required due to the pre-existence of certain interfaces, protocols or restrictions imposed by the technology target of the development. Portability desires are constraint requirements as well, and so are target devices or specific hardware to be used.

Following the guidelines from the ESA standard (aiming not to forget any important aspect of the product specification), User Requirements have to include a variety of attributes, which apply to the definition of every requirement [68].
Since there are two types of User Requirements, we will use identifiers following the expression: **UR-XX-YY**
XX will show the value **CA** for capability requirements or **CO** for constraint requirements.
YY is a two digit counter for user requirements. The counter starts from one for both types of user requirements, hence, two different identifiers with the same digit YY value can be found, namely UR-CA-01 and UR-CO-01.
Identifiers are of chief importance for traceability of the requirements along this document (one instance could be the System Requirements specification, which takes URs as inputs) and the development process. A traceability matrix plotting User Requirements against Software Requirements can be found in Section 3.4.

In addition to the identifier, the following attributes will be included:

- **Need**: specifies the level of necessity for the fulfilment of a requirement, according to its importance regarding achieving the purpose goals from the general description of the system. There are three possible values for this attribute: *Essential*, *Desirable* and *Optional*.

- **Priority**: specifies the urgency of fulfilment of a requirement, its review or creation of lower level requirements. It is a very capital aspect of requirements owing to the fact that such tags allow for a better scheduling and subdivision of tasks once the project has started. There are three priority levels for this project: *Low*, *Medium* and *High*.

- **Stability**: allows to label requirements that are prompt to be changed in the future, so that they can be more easily identified. The two possible values for this attribute are *Stable* or *Unstable*. Most requirements shall not be changing, hence, they will be labelled as *Stable*.

- **Source**: Corresponds to the stakeholder, document or group of users that leaded to the generation of a given requirement. This is often useful to be noted for reviewing purposes, in case some clarifications have to be performed and an appointment or review of the source has to be carried out.

- **Clarity**: Even though requirements are supposed to be written with brevity and precision in mind, this field eases the review process, even in the case of having a single stakeholder involved in the UR phase, and a single developer as it is the case of this dissertation, apart from the tutor (who can take advantage

of this field to criticise the approaches to express certain requirements). There are three values for this field: *Low*, *Acceptable* and *High* clarity.

- **Verifiability**: it is a must for developers to be able to prove that a requirement has been fulfilled, something that can relatively be measured according to how well requirements seem to quantify success, or how similar they look with respect to a clear and precise checklist, avoiding abstract concepts. In other words, user requirements shall be precise enough for them to always be taken into consideration and finally implemented and tested for compliance. This is a binary attribute, thus, it has two possible values: *Verifiable* and *Non-verifiable*.

- **Status**: reflects the degree of advances performed towards the implementation of the requirement. There is a set of values for this attribute: Proposed, Verified (check that it is implemented), Validated (it was checked that a requirement reflects the user needs properly) and Rejected (in case it doesn't apply anymore).

- **Target build**: limits the requirements to a programmed delivery of the ones specified within the Planning part the project, within Section 6.0.2. Two builds are specified in such section; namely **Bare Bones Build** and **Modest Build**.

A table template that comprises all the aforementioned attributes has been created. The template shows the possible values for certain fields and inline clarifications in green colour to ease the understanding of independent fields and user requirements as a whole.

Name	UR-XX-YY
Need	Essential **|** Desirable **|** Optional
Clarity	Low **|** Acceptable **|** High
Priority	Low **|** Medium **|** High
Stability	Unstable **|** Stable
Verifiability	Non-verifiable **|** Verifiable
Source	i.e. Alejandro Rey
Target build	Bare Bones Build **|** Modest Build
Status	Proposed **|** Validated **|** Verified **|** Rejected
Definition	

Table 3.1: Template for user requirements

3.2.1 Capability Requirements

The list of actions that the user shall be able to perform will be provided formally by means of this type of requirements, as stated previously.

Name	UR-CA-01
Need	Essential
Clarity	Acceptable
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The user shall be able to trigger a percussion sound by hitting a random surface with the custom MIDI controller (hard enough as to be considered interaction).

Table 3.2: *Capability User Requirement* 01

Name	UR-CA-02
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The user shall be able to trigger more than three different sounds by interacting with the system (more specifically, with the MIDI controller part of the system).

Table 3.3: *Capability User Requirement* 02

Name	UR-CA-03
Need	Essential
Clarity	Acceptable
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Definition	The user shall be able to understand when the system is up and running, ready to receive user input.

Table 3.4: *Capability User Requirement* 03

Name	UR-CA-04
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The user shall be able to use the MIDI Controller part of the system as a general purpose MIDI controller, that is, as input to any DAW (e.g. Cubase), considering limited functionality.

Table 3.5: *Capability User Requirement* 04

Name	UR-CA-05
Need	Essential
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The user shall be able to generate sounds which vary in volume proportionally to the exerted force on the MIDI controller inputs.

Table 3.6: *Capability User Requirement* 05

Name	UR-CA-06
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The user shall be able to produce sounds from a library of predefined and fixed sounds.

Table 3.7: *Capability User Requirement* 06

Name	UR-CA-07
Need	Essential
Clarity	Acceptable
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The user shall be presented with a visual representation of the instrument to model.

Table 3.8: *Capability User Requirement* 07

Name	UR-CA-08
Need	Essential
Clarity	Acceptable
Priority	Medium
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The user shall be able to understand how sounds relate to their interactions by alterations in objects in virtual space.

Table 3.9: *Capability User Requirement* 08

Name	UR-CA-09
Need	Essential
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The user shall be able to use a general purpose MIDI controller to interact with the system's Desktop application.

Table 3.10: *Capability User Requirement* 09

Name	UR-CA-10
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Modest Build
Status	Validated
Definition	The user shall be able to perform up to 4 (simultaneous) interactions without the system missing the input.

Table 3.11: *Capability User Requirement* 10

Name	UR-CA-11
Need	Essential
Clarity	Acceptable
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Modest Build
Status	Validated
Definition	The user shall be able to leverage previously gained skills regarding real-world drumming practice to interact with the system, they shall be able to interact similarly to the way they would with a real-world drumset.

Table 3.12: *Capability User Requirement* 11

Name	UR-CA-12
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Modest Build
Status	Validated
Definition	The user shall be able to trigger two sequential sounds by performing two consecutive interactions separated by a maximum amount of time of 68ms (maximum separation between two sixteenth notes at 220bmp) without the system missing any input.

Table 3.13: *Capability User Requirement* 12

Name	UR-CA-13
Need	Desirable
Clarity	Acceptable
Priority	Low
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Modest Build
Status	Validated
Definition	The user shall be able to alter the way sounds are mapped to HW interactions from the application.

Table 3.14: *Capability User Requirement* 13

Name	UR-CA-14
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Modest Build
Status	Validated
Definition	The user shall be able to adjust the volume of the whole instrument from the application.

Table 3.15: *Capability User Requirement* 14

Name	UR-CA-15
Need	Desirable
Clarity	Acceptable
Priority	Low
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Modest Build
Status	Validated
Definition	The user shall be able to modify the sounds used for each HW interaction input, choosing different sounds to assign to different parts of the virtual representation of the drumset.

Table 3.16: *Capability User Requirement* 15

Name	UR-CA-16
Need	Desirable
Clarity	Acceptable
Priority	Low
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Modest Build
Status	Validated
Definition	The user shall be able to preview the sounds to be assigned to a drum piece before actually assigning them.

Table 3.17: *Capability User Requirement* 16

3.2.2 Constraint Requirements

The following requirements impose constraints on the solution that is to be provided in this document, restricting how certain objectives shall be achieved.

Name	UR-C0-01
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The MIDI controller shall be built in an Arduino-compatible board in order to ease prototyping and development.

Table 3.18: *Constraint User Requirement* 01

Name	UR-C0-02
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The MIDI protocol shall be used as upper-application level protocol for the logical transmission and management of out-flow and in-flow of messages between subsystems.

Table 3.19: *Constraint User Requirement* 02

Name	UR-C0-03
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	MIDI mappings shall support General MIDI assignments.

Table 3.20: *Constraint User Requirement* 03

Name	UR-C0-04
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The MIDI controller subsystem shall be compatible with default mappings within Addictive Drums 2.

Table 3.21: *Constraint User Requirement* 04

Name	UR-C0-05
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The interface model between MIDI controller and sound trigger need not be self-contained (auxiliary bridge programs may be used).

Table 3.22: *Constraint User Requirement* 05

Name	UR-C0-06
Need	Essential
Clarity	Acceptable
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The user shall be able to understand the effects of its physical interaction by looking at the MIDI controller interface.

Table 3.23: *Constraint User Requirement* 06

Name	UR-C0-07
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	NoteOn messages within the general MIDI specification shall be supported.

Table 3.24: *Constraint User Requirement* 07

Name	UR-C0-08
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The sound-generating part of the system shall be built using the Unity Game Engine Personal Edition, which reduces the cost of the development as it is free of charge.

Table 3.25: *Constraint User Requirement* 08

Name	UR-C0-09
Need	Essential
Clarity	Acceptable
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The user shall be shown virtual imagery to map interaction with visual and auditory output from the Unity-based application.

Table 3.26: *Constraint User Requirement* 09

Name	UR-C0-10
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Bare Bones Build
Status	Validated
Definition	The application shall be targeted at machines running Windows.

Table 3.27: *Constraint User Requirement* 10

Name	UR-C0-11
Need	Desirable
Clarity	Acceptable
Priority	Medium
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Modest Build
Status	Validated
Definition	The system shall resemble the physical interaction with a real-world drumset.

Table 3.28: *Constraint User Requirement* 11

Name	UR-C0-12
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Modest Build
Status	Validated
Definition	MIDI CC messages included in the MIDI protocol specification shall be supported by the MIDI Controller.

Table 3.29: *Constraint User Requirement* 12

Name	UR-C0-13
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Modest Build
Status	Validated
Definition	The MIDI controller's overall dimensions shall be smaller than those of an average 5-piece drumkit (see [69]) .

Table 3.30: *Constraint User Requirement* 13

Name	UR-C0-14
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Modest Build
Status	Validated
Definition	The MIDI controller's overall weight shall be smaller than 4kg so that it is easily portable.

Table 3.31: *Constraint User Requirement* 14

Name	UR-C0-15
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Modest Build
Status	Validated
Definition	The user shall be presented with a virtual 3D model of a drumset within the application.

Table 3.32: *Constraint User Requirement* 15

Name	UR-C0-16
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
Source	Alejandro Rey
Target build	Modest Build
Status	Validated
Definition	The user shall be able to explore the virtual environment, moving around the camera in order to see the complete drum-set towards which the camera shall be placed.

Table 3.33: *Constraint User Requirement* 16

3.3 System Requirements

This section covers the second phase of elicitation, translating User Requirements into System Requirements, going through a previous step involving the creation of Use Cases. The SR phase, often referred to as "problem analysis phase", consists in building a logical model of the system to be built, meaning a simplified, abstract description of the system that can be used as input for the design phase. Systems requirements will show what the system must do and progressively go further in the level of precision on describing the system at hand. The approach to perform this specification, used to reason about the software in broad terms, is formally known as **Functional decomposition** and a guide to its application can be found in PSS-05-03 of [68]; "Guide for the software requirements definition phase" .

3.3.1 Use cases

A Use case diagram is the primary form of system/software requirements for a new software program under development. These define the possible interactions between the actors (users, external entities or systems, etc) and the system at hand. Therefore, is a simplified view to show the desired capabilities and high-level processes involved in each functionality provided to the user.
You can spot two types of relationships between processes (formally named Use cases, ovals in the picture), *include* and *extend*. An *include* relationship means that a use case or process is inherent to the execution of another use case ,that is, if the base one runs the other does. In contrast, an *extend* relationship implies an optional use case when another happens, that is, depending on the user actions or specific situation such use case may be executed or not.
For *extend* relationships you can see the arrowhead points to the use case to which one is complementing, whereas in an *include* relationship the arrowhead points towards use cases that could be though as sub-processes of others.

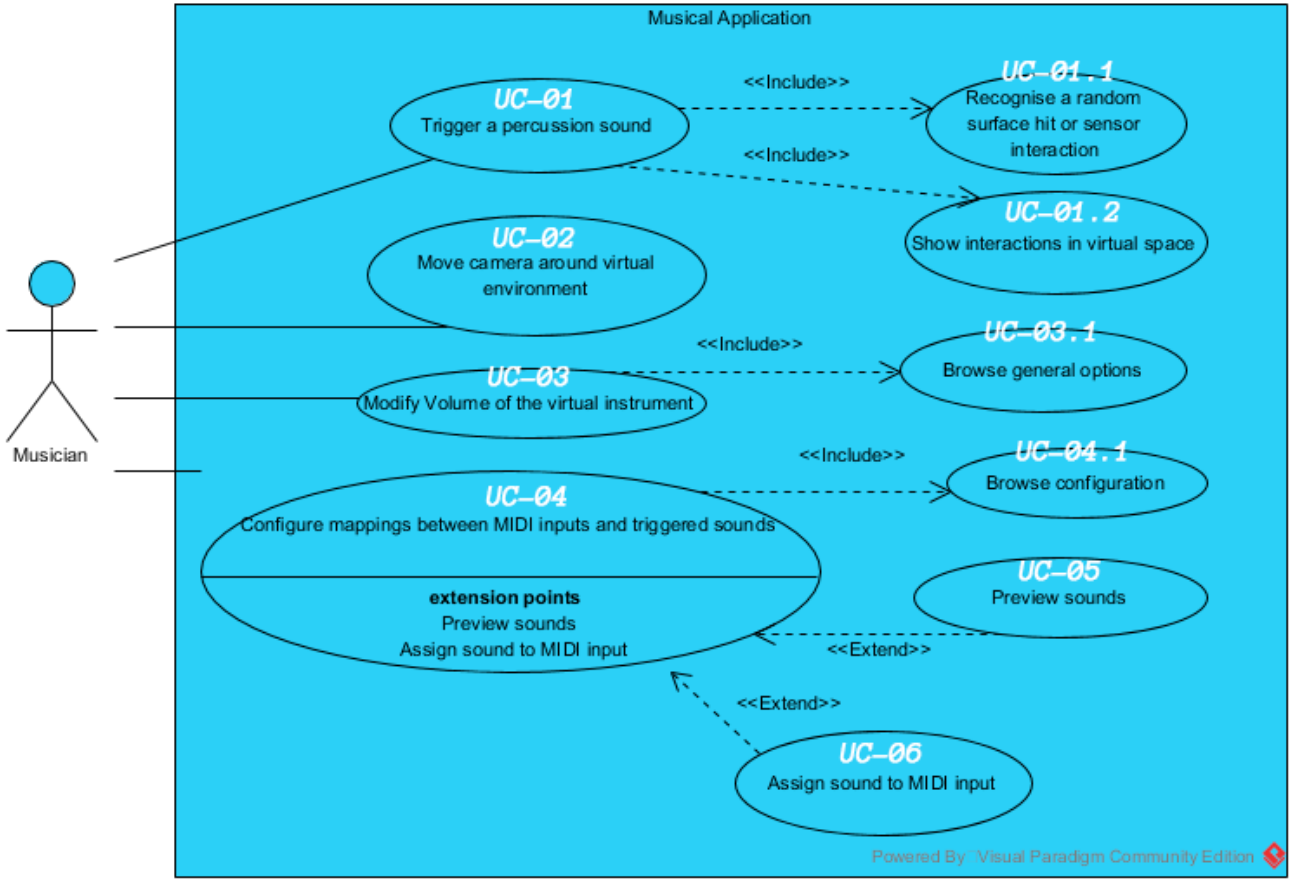


Figure 3.6: Use Case Diagram

3.3.2 Software Requirements Specification

In the same way we created the User Requirements, a set of lower requirements are to be enumerated to convert from user expectations and views to what the system shall do and the constraints imposed to the system design. Therefore, similarly to the templates and categorisation in Section 3.2, we will provide a new one for this kind of requirements.
System requirements can be split into two categories, Functional Requirements and Non-functional requirements, the differences between these two categories are briefly summarised below:

- **Functional requirements**: define functions required to accomplish the objectives of the system. In other words, they will detail the set of operations the software shall be able to perform.
- **Non-functional requirement**: define constraints imposed over the performance, the shape of the system, the interfaces it will be interacting with, the environmental conditions under which the system must keep working, reliability issues that must be taken into account, etc. These constitute a less fuzzy representation of the constraint requirements described in the UR phase. Regarding the type of constraints they impose, they can be categorised according to NASA guidelines [70] into: **Performance**, **Constraints**, **Interface**, **Environmental**, **Human Factors**, **Reliability**, **Verification** and **Safety** requirements.

Software requirements will expand upon the attributes specified for user requirements to handle traceability better, so the following template has been created to hold all the required information about the requirements:
Since there are two types of Software Requirements, we will use identifiers following the expression: **SR-XX-YY**
XX will show the value **FR** for Functional requirements or **CO** for Non-functional requirements (also known as Constraint requirements).
YY is a two digit counter for system requirements. The counter starts from one for both types of System requirements, so once more, two different identifiers with the same digit YY value can be found (e.g. UR-FR-01 and UR-NF-01).
A traceability matrix plotting User Requirements against Software Requirements can be found in Section 3.8 .

In addition to the identifier, the following attributes will be included:

- **Need**: specifies the level of necessity for the fulfilment of a requirement, according to its importance to achieve the purpose goals from the general description of the system. There are three possible values for this attribute: *Essential*, *Desirable* and *Optional*.
- **Priority**: specifies the urgency of fulfilment of a requirement, its review or creation of lower level requirements (if any). There are three priority levels for this project: *Low*, *Medium* and *High*.

- **Stability**: allows to label requirements that are prompt to be changed in the future, so that they can be more easily identified. The two possible values for this attribute are *Stable* or *Unstable*. Most requirements should not be changing, so they will be labelled as *Stable*.
- **UR Source**: Includes a non-empty set of 1 or more User requirements that originated the creation of each specific System requirement, which guarantees that the developer focuses on providing the essential functionality the user asked for, neither more nor less.
- **UC Source**: States the associated Use Case (see Section 3.3.1) to which the software requirement is associated.
- **Clarity**: States the level of ambiguity a requirement presents. There are three values for this field: *Low*, *Acceptable* and *High* clarity.
- **Verifiability**: States whether the requirement can be formally verified, that its, its fulfilment can be claimed to be successful. This is a binary attribute, thus, it has two possible values: *Verifiable* and *Non-verifiable*.
- **Status**: reflects the degree of advances performed towards the implementation of the requirement. There is a set of values for this attribute: Proposed, Verified (check that it is implemented), Validated (it was checked that a requirement reflects the user needs properly) and Rejected (in case it does not apply anymore).
- **Target build**: limits the requirements to a programmed delivery of the ones specified within the Planning part the project, within Section 6.0.2. Two builds are specified in such section; namely **Bare Bones Build** and **Modest Build**.

Two table templates that comprise all the aforementioned attributes have been created. The templates show the possible values for certain fields and inline clarifications to ease the understanding of independent fields and user requirements as a whole. Note that the Non-functional requirements will be further categorised into the sub-types mentioned earlier in this section, to show such taxonomy, these requirements will present an extra field, called Sub-type, highlighted in the template with an orange colour.

Functional Requirement	
Name	SR-FR-YY
Need	Essential | Desirable | Optional
Clarity	Low | Acceptable | High
Priority	Low | Medium | High
Stability	Unstable | Stable
Verifiability	Non-verifiable | Verifiable
UR Source	i.e. UR-CA-01
UC Source	i.e. UC-01
Target build	Bare Bones Build | Modest Build | Future Work
Status	Proposed | Validated | Verified | Rejected
Definition	

Table 3.34: Template for Functional Software requirements

Non-functional Requirement	
Name	SR-NF-YY
Subtype	Performance | Constraints | Interface | Environmental | Human Factors | Reliability | Safety | Verification
Need	Essential | Desirable | Optional
Clarity	Low | Acceptable | High
Priority	Low | Medium | High
Stability	Unstable | Stable
Verifiability	Non-verifiable | Verifiable
UR Source	i.e. UR-CA-01
UC Source	i.e. UC-01
Target build	Bare Bones Build | Modest Build | Future Work
Status	Proposed | Validated | Verified | Rejected
Definition	

Table 3.35: Template for Software requirements

3.3.3 Functional Requirements

This section covers the formal definition of software requirements that correspond to functionality the final application is to implement, as defined earlier in this document. These requirements set up the baseline for design and development and may be referenced in subsequent sections.
Note that functional requirements are easily identified by means of the colour of the table header, which has got a light green background and white text.
Requirements present here will be verified in the Evaluation Section 5.2 and consequently, the status for most of the requirements mentioned herein is expected to be Validated at most, which simply involves properly reflecting the needs of the user.

Functional Requirement	
Name	SR-FR-01
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-01
UC Source	UC-01
Target build	Bare Bones Build
Status	Verified
Definition	The system shall play back a sound that is associated with a specific interaction from the user.

Table 3.36: SR-FR-01

Functional Requirement	
Name	SR-FR-02
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-01
UC Source	UC-01
Target build	Bare Bones Build
Status	Verified
Definition	The system shall continuously monitor and process input from the user when a sound is configured as to be played in response.

Table 3.37: SR-FR-02

Functional Requirement	
Name	SR-FR-03
Need	Essential
Clarity	Acceptable
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-03
UC Source	UC-01
Target build	Bare Bones Build
Status	Verified
Definition	The MIDI controller subsystem shall provide with output feedback to the user, so that the user can understand the state of the MIDI controller subsystem.

Table 3.38: SR-FR-03

Functional Requirement	
Name	SR-FR-04
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-02
UC Source	UC-01
Target build	Bare Bones Build
Status	Verified
Definition	The system shall be able to generate more than three different sounds to be triggered as response to different user input gestures.

Table 3.39: SR-FR-04

Functional Requirement	
Name	SR-FR-05
Need	Essential
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-04
UC Source	UC-01
Target build	Bare Bones Build
Status	Verified
Definition	The MIDI Controller subsystem shall send MIDI Messages to the Sound Generation Subsystem.

Table 3.40: SR-FR-05

Functional Requirement	
Name	SR-FR-06
Need	Essential
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-05
UC Source	UC-01
Target build	Bare Bones Build
Status	Verified
Definition	The MIDI Controller subsystem shall create MIDI messages whose velocity depends on the strength of a hit in order to notify the Sound generation subsystem about it.

Table 3.41: SR-FR-06

Functional Requirement	
Name	SR-FR-07
Need	Essential
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-05
UC Source	UC-01
Target build	Bare Bones Build
Status	Verified
Definition	The system shall generate sounds that vary in volume proportionally to the provided MIDI velocity, extracted by the MIDI subsystem and associated to the force exerted on some sensor.

Table 3.42: SR-FR-07

Functional Requirement	
Name	SR-FR-08
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-06
UC Source	UC-01
Target build	Bare Bones Build
Status	Verified
Definition	The system shall load sounds from a set of predefined sounds within the computer the Sound generation subsystem is being run into.

Table 3.43: SR-FR-08

Functional Requirement	
Name	SR-FR-09
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-06
UC Source	UC-01
Target build	Bare Bones Build
Status	Verified
Definition	The system shall assign sounds to MIDI Number inputs by default.

Table 3.44: SR-FR-09

Functional Requirement	
Name	SR-FR-10
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-08
UC Source	UC-01
Target build	Bare Bones Build
Status	Verified
Definition	The system shall inform the user about which drum piece has been virtually hit by means of visualisation mechanisms.

Table 3.45: SR-FR-10

Functional Requirement	
Name	SR-FR-11
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-13
UC Source	UC-04
Target build	Modest Build
Status	Validated
Definition	The Sound Generation subsystem shall allow configuration of how sounds are mapped to specific interactions with the MIDI Controller subsystem.

Table 3.46: SR-FR-11

Functional Requirement	
Name	SR-FR-12
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-13
UC Source	UC-04
Target build	Modest Build
Status	Validated
Definition	The Sound Generator subsystem shall provide a graphical user interface to be used for configuration by the user of the system.

Table 3.47: SR-FR-12

Functional Requirement	
Name	SR-FR-13
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-14
UC Source	UC-03
Target build	Modest Build
Status	Validated
Definition	The Sound Generator subsystem shall allow the user to control the overall volume generated as output.

Table 3.48: SR-FR-13

Functional Requirement	
Name	SR-FR-14
Need	Desirable
Clarity	High
Priority	Low
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-15
UC Source	UC-06
Target build	Modest Build
Status	Validated
Definition	The system shall allow to modify the sounds assigned to different interactions on the MIDI Controller subsystem.

Table 3.49: SR-FR-14

Functional Requirement	
Name	SR-FR-15
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-07
UC Source	UC-01
Target build	Bare Bones Build
Status	Validated
Definition	The MIDI Controller subsystem shall be able to generate NoteOn messages.

Table 3.50: SR-FR-15

Functional Requirement	
Name	SR-FR-16
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-07
UC Source	UC-01
Target build	Bare Bones Build
Status	validated
Definition	The Sound generation subsystem shall be able to process and extract the encoded information from standard NoteOn MIDI messages.

Table 3.51: SR-FR-16

Functional Requirement	
Name	SR-FR-17
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-12
UC Source	UC-01
Target build	Modest Build
Status	Validated
Definition	The MIDI Controller subsystem shall be able to generate CC messages.

Table 3.52: SR-FR-17

Functional Requirement	
Name	SR-FR-18
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-12
UC Source	UC-01
Target build	Modest Build
Status	Validated
Definition	The Sound generation subsystem shall be able to process and extract the encoded information from standard CC MIDI messages.

Table 3.53: SR-FR-18

Functional Requirement	
Name	SR-FR-19
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-16
UC Source	UC-02
Target build	Modest Build
Status	Validated
Definition	The Unity camera's movement shall be enabled so that users can navigate virtual space around the 3D representation of the drumset.

Table 3.54: SR-FR-19

Functional Requirement	
Name	SR-FR-20
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-16
UC Source	UC-05
Target build	Modest Build
Status	Validated
Definition	The system shall allow the user to pre-listen sounds before they get to assign them to a specific MIDI subsystem input.

Table 3.55: SR-FR-20

Functional Requirement	
Name	SR-FR-21
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-04
UC Source	UC-01
Target build	Bare Bones Build
Status	Validated
Definition	The MIDI Controller subsystem shall be able to send NoteOn messages compliant with the Standard MIDI Specification.

Table 3.56: SR-FR-21

Functional Requirement	
Name	SR-FR-22
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-04
UC Source	UC-01
Target build	Bare Bones Build
Status	Validated
Definition	The MIDI Controller subsystem shall be able to send Control Change (CC) messages compliant with the Standard MIDI Specification.

Table 3.57: SR-FR-22

3.3.4 Non-Functional Requirements

This subsection enumerates all the Non-functional requirements constraining the design and development of the system, further categorising them according to the aspect of the system they cover.
Since User Cases show the main actions the user shall be able to perform in the system, these have no direct relation with non-functional requirements.Because of that, Non-functional requirements have the UR Source field tagged as N/A (Not Applicable).

Non-functional Requirement	
Name	SR-NF-01
Subtype	Verification
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-01
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The system shall generate sounds only from direct input of the user, that is, not arbitrarily.

Table 3.58: SR-NF-01

Non-functional Requirement	
Name	SR-NF-02
Subtype	Constraints
Need	Essential
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-01
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The system shall only generate sounds when a sound is being previewed or when direct interaction via custom hardware is performed.

Table 3.59: SR-NF-02

Non-functional Requirement	
Name	SR-NF-03
Subtype	Constraints
Need	Essential
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-03
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The system shall provide output feedback to the user using hardware actuators.

Table 3.60: SR-NF-03

Non-functional Requirement	
Name	SR-NF-04
Subtype	Interface
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-03
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The connection between the MIDI Controller subsystem and the Sound generating subsystem shall be performed by means of USB.

Table 3.61: SR-NF-04

Non-functional Requirement	
Name	SR-NF-05
Subtype	Verification
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-04
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The MIDI Controller subsystem shall be tested using Cubase as sequencer to show interoperability.

Table 3.62: SR-NF-05

Non-functional Requirement	
Name	SR-NF-06
Subtype	Constraints
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	i.e. UR-CA-07
UC Source	N/A
Target build	Bare Bones Build
Status	Verified
Definition	The system shall show via a display a set of 3D objects representing different drum pieces.

Table 3.63: SR-NF-06

Non-functional Requirement	
Name	SR-NF-07
Subtype	Constraints
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-09
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The Sound generation subsystem shall receive data from all MIDI ports within the Desktop computer the program is running on.

Table 3.64: SR-NF-07

Non-functional Requirement	
Name	SR-NF-08
Subtype	Verification
Need	Essential
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-09
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The System shall be testable using a general purpose MIDI controller, often in the shape of a piano keyboard.

Table 3.65: SR-NF-08

Non-functional Requirement	
Name	SR-NF-09
Subtype	Performance
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-10
UC Source	N/A
Target build	Modest Build
Status	Validated
Definition	The system shall be able to track up to 4 simultaneous interactions with the MIDI controller sensors, meaning by tracking allowing a correct execution without data loses 80% of the times (computed over a sample of 50 trials).

Table 3.66: SR-NF-09

Non-functional Requirement	
Name	SR-NF-10
Subtype	Constraints
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-11
UC Source	N/A
Target build	Modest Build
Status	Validated
Definition	The system shall provide input mechanisms that involve hitting or pressing.

Table 3.67: SR-NF-10

Non-functional Requirement	
Name	SR-NF-11
Subtype	Performance
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CA-12
UC Source	N/A
Target build	Modest Build
Status	Validated
Definition	The MIDI Controller subsystem shall be able to recognise two consecutive interactions separated by a maximum amount of time of 68ms, to guarantee the MIDI Controller subsystem is still usable at high speeds.

Table 3.68: SR-NF-11

Non-functional Requirement	
Name	SR-NF-12
Subtype	Constraints
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-01
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The MIDI Controller subsystem shall be built using an Arduino-compatible board with actuators.

Table 3.69: SR-NF-12

Non-functional Requirement	
Name	SR-NF-13
Subtype	Interface
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-02
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The system shall use the MIDI communication protocol to transmit data between the MIDI Controller subsystem and the Sound Generation Subsystem.

Table 3.70: SR-NF-13

Non-functional Requirement	
Name	SR-NF-14
Subtype	Constraints
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-03
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The MIDI Controller subsystem shall generate output MIDI Messages using the note values in GM Standard.

Table 3.71: SR-NF-14

Non-functional Requirement	
Name	SR-NF-15
Subtype	Constratints
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-04
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The MIDI Controller messages shall work in conjunction with Addictive Drums 2.

Table 3.72: SR-NF-15

Non-functional Requirement	
Name	SR-NF-16
Subtype	Verification
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-04
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The MIDI Controller subsystem shall be tested using Addictive Drums 2.

Table 3.73: SR-NF-16

Non-functional Requirement	
Name	SR-NF-17
Subtype	Constraints
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-05
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The system need not be self-contained.

Table 3.74: SR-NF-17

Non-functional Requirement	
Name	SR-NF-18
Subtype	Constraints
Need	Essential
Clarity	Acceptable
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-06
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The MIDI Controller subsystem shall show information about the state of the subsystem by means of external actuators.

Table 3.75: SR-NF-18

Non-functional Requirement	
Name	SR-NF-19
Subtype	Constraints
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-08
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The Sound generator subsystem shall be developed based on the Unity Game Engine Personal Edition.

Table 3.76: SR-NF-19

Non-functional Requirement	
Name	SR-NF-20
Subtype	Constraints
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-09
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The Sound generator subsystem shall display virtual objects to represent different drum pieces.

Table 3.77: SR-NF-20

Non-functional Requirement	
Name	SR-NF-21
Subtype	Constraints
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-10
UC Source	N/A
Target build	Bare Bones Build
Status	Validated
Definition	The Sound generator subsystem shall run in Windows Machines running Windows 10.

Table 3.78: SR-NF-21

Non-functional Requirement	
Name	SR-NF-22
Subtype	Constraints
Need	Desirable
Clarity	Acceptable
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-11
UC Source	N/A
Target build	Modest Build
Status	Validated
Definition	The system shall imitate the interaction paradigm present when interacting with a real drum-set.

Table 3.79: SR-NF-22

Non-functional Requirement	
Name	SR-NF-23
Subtype	Constraints
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-13
UC Source	N/A
Target build	Modest Build
Status	Validated
Definition	The MIDI Controller subsystem's dimensions shall be smaller than (width = 84 cm x deep = 72 cm)

Table 3.80: SR-NF-23

Non-functional Requirement	
Name	SR-NF-24
Subtype	Constraints
Need	Desirable
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	i.e. UR-CO-14
UC Source	N/A
Target build	Modest Build
Status	Validated
Definition	The MIDI controller subsystem weight shall be smaller than 4kg .

Table 3.81: SR-NF-24

Non-functional Requirement	
Name	SR-NF-25
Subtype	Constraints
Need	Essential
Clarity	High
Priority	High
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-15
UC Source	N/A
Target build	Modest Build
Status	Validated
Definition	The Sound generating subsystem shall display a virtual model of a drumset.

Table 3.82: SR-NF-25

Non-functional Requirement	
Name	SR-NF-26
Subtype	Constraints
Need	Essential
Clarity	High
Priority	Medium
Stability	Stable
Verifiability	Verifiable
UR Source	UR-CO-16
UC Source	N/A
Target build	Modest Build
Status	Validated
Definition	The Sound generating subsystem allow configuration by overlaying a configuration menu within the screen of the computer the Sound generating subsystem is running on.

Table 3.83: SR-NF-26

3.3.5 System Requirements Specification

3.4 Traceability Matrix

In this section, two tables are shown, used as a measure for traceability of the requirements. These show how different levels of application requirements map each other meaning their source is a higher level requirement specification. Figure 3.8 shows which User requirements gave birth to System Requirements, assuring the

fact that all of them will be taken into account for the design part. In contrast Figure 3.7 shows how the main use actions the user is allowed to perform map to functional requirements within the System Requirements Specification Section.

	UC-01	UC-02	UC-03	UC-04	UC-05	UC-06
SR-FR-01	X					
SR-FR-02	x					
SR-FR-03						
SR-FR-04	x					
SR-FR-05						
SR-FR-04						
SR-FR-05	x					
SR-FR-06	x					
SR-FR-07	x					
SR-FR-08	x					
SR-FR-09	x					
SR-FR-10	x					
SR-FR-11				x		
SR-FR-12				x		
SR-FR-13			x			
SR-FR-14						x
SR-FR-15	x					
SR-FR-16	x					
SR-FR-17	x					
SR-FR-18	x					
SR-FR-19		x				
SR-FR-20					x	
SR-FR-21	x					
SR-FR-22	x					
SR-NF-01						
SR-NF-02						
SR-NF-03						
SR-NF-04						
SR-NF-05						
SR-NF-06						
SR-NF-07						
SR-NF-08						
SR-NF-09						
SR-NF-10						
SR-NF-11						
SR-NF-12						
SR-NF-13						
SR-NF-14						
SR-NF-15						
SR-NF-16						
SR-NF-17						
SR-NF-18						
SR-NF-19						
SR-NF-20						
SR-NF-21						
SR-NF-22						
SR-NF-23						
SR-NF-24						
SR-NF-25						
SR-NF-27						
	UC-01	UC-02	UC-03	UC-04	UC-05	UC-06
	Use Cases					

Figure 3.7: Use Cases vs System Requirements traceability matrix

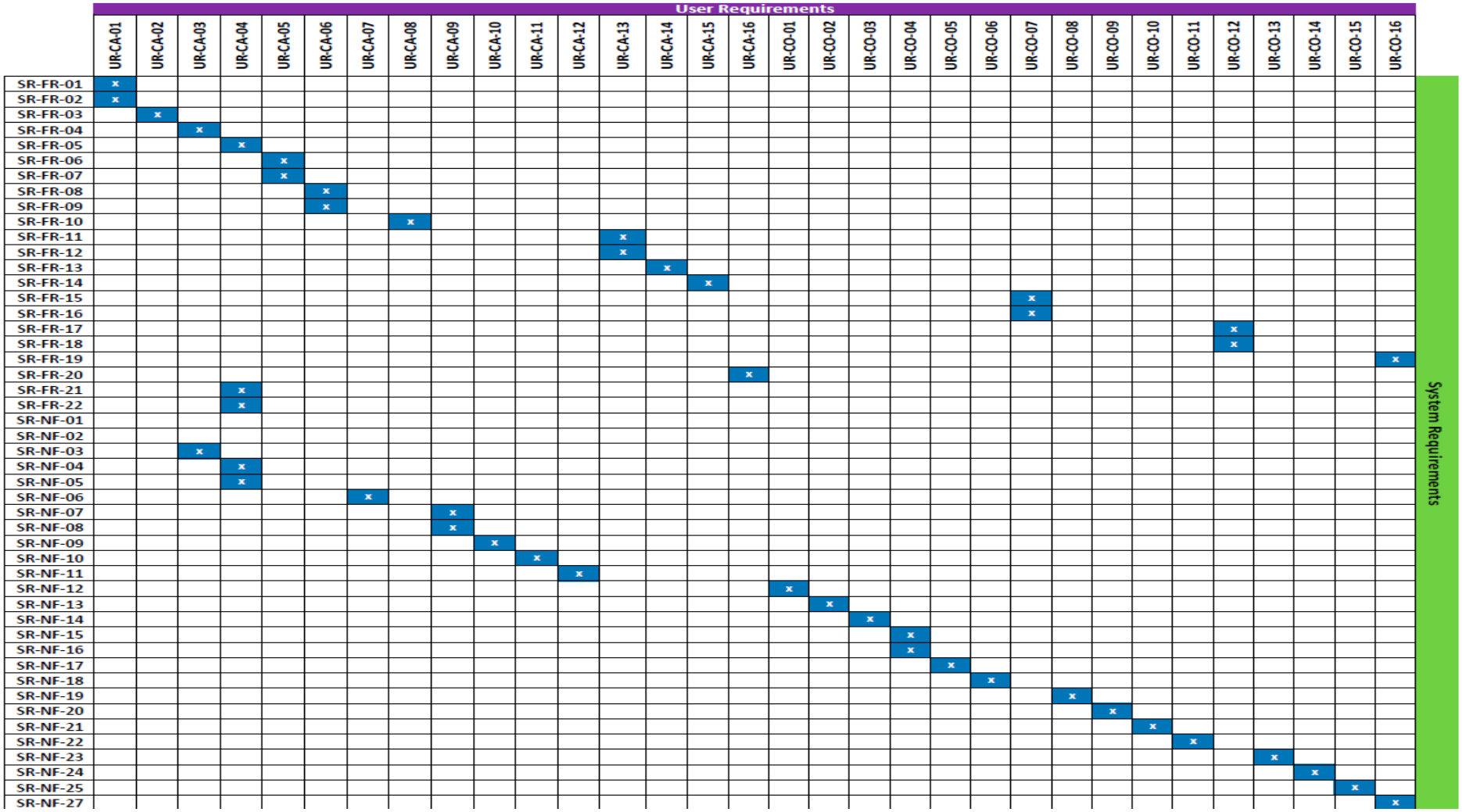


Figure 3.8: System Requirements vs User Requirements traceability matrix

Chapter 4

Design of the Solution

This chapter covers the architectural design and implementation concerns regarding ***DrumVR*** , the reasoning behind the selection of certain technologies, the reflections on the problem to be solved by means of the created system, the hardware and algorithms employed to satisfy the requirements for the system as well as auxiliary comments which discuss how to solve some of the most common problems found along the development of this project.

4.1 Evaluation of complexity and design alternatives

This section covers the design paths that were weighed up before committing to the final solution for the system described in this document, evaluating each design possibility independently and explaining the arguments upon which each final decision was made.

As Jordà spotted in [50], VMIs comprise two clearly differentiated components: a **gesture controller** (a piece of equipment that a player uses to control how sounds generated or stored in a computer will be released) and a **sound generator** (which is responsible for triggering or generating sound in real-time). This approach to VMI design was standardised right after the appearance of MIDI and allowed for independent development of both ends of a VMI. It enhanced flexibility as well, as MIDI allows to control any sound generator using any device that implements MIDI communication. This new approach to VMI design was known as 'splitting the instrument chain'.
Since the aforementioned high-level architecture is unavoidably applied to the present project, the discussion about design paths will be split according to the three major concerns of the project: the MIDI Controller subsystem (Section 4.1.1), the Sound generator subsystem (Section 4.1.2) and the integration aspects provided to interconnect both (Section 4.1.3).

4.1.1 MIDI Controller Subsystem alternatives

The system is expected to implement a MIDI controller with custom hardware whose portability is high and which ultimate goal is to generate sounds according to some input from the user, in the shape of hits, the interaction form drummers are accustomed to. From the System requirements, several major approaches to the solution of this subsystem were evaluated, with three outstanding possibilities:

- **Creating a custom DIY electronic drumset based on pads**: One of the examined approaches to the problem involved creating a conventionally-shaped drum MIDI controller, which showed a main advantage regarding known feasibility and easy estimation of development time; but on the contrary, it offered no real innovation on the interaction paradigm and neither allowed for an improvement in portability of the system nor provided an enhancement in flexibility. Thus, this approach was soon discarded due to its numerous limitations.

- **Delegating input recognition to XR subsystem**: an early approach that was looked into consisted in using tracking visual recognition mechanisms in the market to trigger sounds directly from the computer-side application, independent of the specific target device used at the end of the development process. During a brief feasibility study, the Vuforia library [1] (integrated with Unity) was explored shortly, it included features that easily allowed for presentation of holograms in real space by means of image targets (see an example in Figure 4.1), which could be used as a great way for users to arrange their drumkit pieces in AR space to taste. The Vuforia library is said to speed up the development of VR/AR applications by adding computer vision features for UWP among other platforms. Therefore, this library was suitable for Hololens development but still, the main disadvantage of this approach laid in how well fast hand and foot interactions could be handled by computer vision. It was also considered the fact that due to the limitations in performance of the library, the final interaction model would probably need to differ a lot from the approach used for interacting with a real drumset and therefore, users would not be able to leverage their existing skills.
  Moreover, learning how to use this library seemed like a real challenge fromt he programmer's perspective, and thus, this approach to design could be postponed to later iterations of the system. Hololens examples provided in the Mixed Reality Toolkit by Microsoft [2] also showed feasibility of interaction with projected 3D models by means of combined gaze and gesture input, but that was still a problem because its performance was limited. Thus, this approach was dismissed along with the previous one.

- **Creating a custom MIDI controller whose interaction was built based on the drumsticks and the leverage of all four limbs of the user**: inspired by systems such as Aerodrums [48] and Freedrum (this one discovered

[1] https://developer.vuforia.com/
[2] see https://github.com/microsoft/MixedRealityToolkit-Unity

Figure 4.1: Example of Hologram projected using image target; via https://library.vuforia.com/articles/Training/Image-Target-Guide

by the end of the project[3]), one of the ideas behind the interaction paradigm to be supported involved integrating the sensors to be used to capture user input into a pair of drumsticks and pedals, so that it could be possible to avoid the necessity of carrying a lot of stuff with you to play MIDI drums, increasing portability. Since similar projects existed, as the ones mentioned, the feasibility was guaranteed and according to research, it was a relatively easy approach to use, considering the great amount of DIY projects regarding MIDI that can be found over the Internet. In addition, this approach might be better for a novice developer of virtual musical instruments and embedded-systems in this case, so since the learning resources were plentiful, then less trouble was estimated to be found along the development process.
Thus, a MIDI controller of these characteristics was set as a goal to be achieved in this dissertation.

4.1.2 Software subsystem (Sound Generator) alternatives

Regarding the sound generation subsystem, which had as main desired capability providing with feedback to the user in the form of audio and visuals; many different possible alternatives appeared as solutions for the aforementioned system requirements:

- **Creating a standard UWP application from scratch**: an early possibility taken into account involve creating a Standard Microsoft application to represent and project 3D items in visual space, which due to the focus on 3D imagery seemed hard to do from scratch, without reinventing the wheel provided by game engine environments such as Unity, Unreal and so forth. The fact that so much work and examples were available for other technologies along with the similarities of those programming paradigms with respect to

[3]see https://www.freedrum.rocks/

the prior knowledge of the author of this dissertation were the key point to abandon this a priory costly approach, which differed widely from the way programs were built in Linux, and which feasibility in a timely schedule was unlikely to be guaranteed. In short, this approach was discarded due to the huge amount of work and expertise that is supposed to be owned in order to provide an actual working system with the expected characteristics.

- **Creating a Unity-based videogame program**: aiming at a fast development and prototyping of the system, using a game engine as basis for visuals on our system was proven to be possible, given the nature of videogames and its similarities with respect to the proposed system, in terms of navigation and expected representation of the drumset as such within the virtual space. Additionally, Unity integrates with AR/VR devices and supports building application targeted at an heterogeneous set of platforms with minimum effort, so constituted the most flexible approach for development. The fact that Hololens tutorials were based upon Unity for development was also a critical factor when deciding this path for development, even though the final product didn't finally target this device due to the limitations discovered during the development process (see Chapter **??** for more details). Furthermore, State of the art VRMIs are known to have used Unity for development, such as *Mixed Reality Keyboard* [43]. For these reasons, the selected pathway for SW-side development involved Unity as starting point.

4.1.3 Integration Mechanisms

As the system is known to have two clearly differentiated subsystems that shall exchange information in order for the VMI to perform its basic functions, some integration mechanisms had to be used to "interconnect the instrument chain". In this section, the thoughts upon which the final design choice is made are explained shortly.

To begin with, one of the challenges inherent to the system to be built was integration between the Hardware part of the system (the MIDI controller) and the visualisation program to run in a desktop computer or AR/VR device.
One of the possibilities regarding integration was providing a self-contained solution for the transformation from Serial USB to MIDI, the creation of a MIDI port for later use in the visualisation application. However, this approach seemed too complex regarding the need to explore the creation of drivers and handling communication between Unity and the Hardware device all on the writer's own, in an environment (the Windows operating system) which is completely new for him, as a computer undergraduate used to Linux.
Since after a relatively thorough exploration, free or open-source programs that precisely covered the system's needs were found, it was decided to abandon the self-contained approach to avoid, once more, reinventing the wheel. It was considered as a possibility to provide with such self-contained functionality later on, as new iterations were performed over the first few prototypes, narrowing this way the

scope of this dissertation.
Next, the reader will be presented with the integration concerns and the chosen solutions for them:

- On the one hand, the first issue consisted in taking the Serial input to the computer, encoding MIDI messages, and converting that stream back to MIDI to finally conveying the MIDI messages through a MIDI port. This is precisely what **Hairless MIDI** enables, acting as a Serial to MIDI Bridge[4]. Using this application, development time could be reduced and successful communication among subsystem was enabled in an easy fashion.

- On the other hand, in order to redirect the newly extracted MIDI from the Serial communication protocol at an arbitrary speed towards a MIDI port, we were required to create a MIDI loopback[5] port, that is, a port that takes data from the computer and redirects it towards another application using a MIDI port.
  An application developed by Tobias Erichsen, called **loopMIDI; visit** [6] enables precisely this, allowing for creation of MIDI ports to use as input to other applications. This project uses a driver created by Tobias too, called virtualMIDI, which also provides a SDK, that could be later explored on an advanced iteration of the project to complete the last ideal goal, the project to be self-contained.

- In addition to these two programs, a Unity add-on was used to support MIDI input in Unity, which reads all the MIDI ports and handles basic MIDI inputs such as NoteOn and CC messages within the MIDI specification. This add-on, called **MIDI Jack** and developed by Keijiro Takahashi eases the development process by providing all the low level functionality of dealing with MIDI ports and allows the author of this dissertation to focus on basic handling of interaction and GUI concerns [72]. However, this tool does not support UWP by the writing of this dissertation, which limits the target device to Desktop Windows and gets in the way with Hololens development for the moment. UWP is in fact the way to go to obtaining MIDI-BLE support using the UWP API, which is a future work objective of the author of this dissertation (see Chapter 9 for more information regarding future work).

Led by these decisions on how to build the system prioritising development time and minimum effort, the subsequent design was greatly influenced by this principle, in the sense that most expectations were reduced to the minimum so that the problem, which complexity was initially unknown due to a poor expertise in the matter, could be finally manageable by a single developer. Only after this process of

[4]Hairless MIDI (see https://projectgus.github.io/hairless-midiserial/) is a cross-platform program developed by Angus Gratton which was compatible with Arduino MIDI Library. This application uses Qt, a cross platform framework that expands the capabilities of the C++ language and generate standard compliant C++ sources to be compiled by any known C++ compiler such as Clang or GCC [71]

[5]loopback: connecting via software the host system to itself

[6]http://www.tobias-erichsen.de/software/loopmidi.html

complexity decomposition, a plan regarding the pathway to follow for development and requirements elicitation could be performed. Simplification of the problem into more manageable and specific goals played a huge role in the completion of this dissertation, which otherwise would probably have been too complex to even approach or build in a limited schedule such as the one situation of the author of this dissertation.
As main consequences of the technology selections discussed we can extract the following conclusions:

- **The AR paradigm is abandoned in the Software side**, going for a simplified and easily achievable functioning whole instead of a set of failed or in-progress attempts in the limited time-span devoted to the dissertation. Hololens targetting is thus left as a future goal, and the AR is to be present in a way within the MIDI Controller, ideally building a controller that can generate different sounds depending on the orientation of the drumsticks.
- The ideal **BLE support for MIDI is** also **abandoned** in both ends of the system, due to the increased complexity this support implies and the limitations in the documentation found when briefly exploring this new feature. Allowing for BLE MIDI would also require (at least) to modify the MIDI Jack add-on to support UWP or finding a different solution to provide such support otherwise. It was considered a much better idea to make the problem easier to manage and expand over an existing solution, which leverages the properties of Incremental delivery approach to the SDLF (Software Development Lifecycle), which is used in the project and covered in detail in 6.0.2.
- Some requirements are implemented by means of the third-party software (specifically, SR-FR-16 and SR-NF-17).

4.2 Architectural Design

The present section covers the design of the system and the improvement of it in the numbered builds detailed in Section **??**, both the *Bare Bones Build* and the *Modest Build*. These are also referenced in the requirements, from which this section is derived and which was used as reference for implementation.
In this section, subsequent iterations on the project are depicted one after the other, covering Hardware and Software-based subsystems independently.

4.2.1 Bare Bones Build

A very basic version of the system is covered in this design phase, providing functionality detailed in system functional requirements and considering constraints imposed by non-functional requirements as well. First, in 4.2.1.1 a high level description of the functionality to be supported will be provided, aiming at giving the user a quick comprehensive grasp of the system to be developed and the objectives to be achieved

in this first build. Next, the Hardware-based subsystem(i.e. the gestural controller) design will be covered in Section 4.2.1.2. Afterwards, the Software-based subsystem will be described in 4.2.1.3.

4.2.1.1 Bare Bones Build Concept

The first build of the project has as target to create a proof of concept system that is able to generate sounds in response to user input gathered by means of sensors that are hooked up to an Arduino-compatible board, and which are to provide basic visualisation on the videogame-side of the application. The main objectives to be achieved in this build are the following:

- **Designing a gesture controller** based on hit sensors that allow to obtain a continuous value; that is, analog sensors which can be used to trigger sounds with varying volume depending on the type of interaction, as required by 3.42. All the required circuitry will be attached to a protoboard in order to avoid soldering and so that we can make connections non-permanent. This subsystem will comply as well with the following lower-level objectives:
  - **Implementing MIDI data output** from the gesture controller supporting at least NoteOn and CC MIDI messages, as required by 3.50 and 3.57.
  - **Including actuators which notify the user about the state of the gesture controller** as required by 3.39. These actuators have been chosen to be simple LEDs, as it will be specified later.
  - **Designing a control algorithm** that continuously polls for user input and converts the input information into MIDI output messages (as specified 3.37), whose velocity value is proportional to the value received as input from the corresponding analog sensors (as stated in 3.42).
- **Designing a sound generator** which is able to to process MIDI input data coming from the gesture controller and provide both basic auditory and visual feedback. This subsystem will comply as well with the lower-level following objectives:
  - **Allowing for triggering of at least 3 different fixed sounds depending on the MIDI note number received**, **using the velocity field of NoteOn messages as a value to modify the volume of the sound sample to be triggered** (the greater the velocity, the higher the volume). Mappings between MIDI numbers and sound shall be fixed and created by default (as extracted from **??**) and sounds shall be extracted from a limited library of .
  - **Creating a simple 3D model representing the drumset in the virtual environment** whose materials are altered when the user virtually hits a specific piece of the simulated drumset (derived from 3.45) .

4.2.1.2 Hardware Design: Arduino-Based MIDI Controller

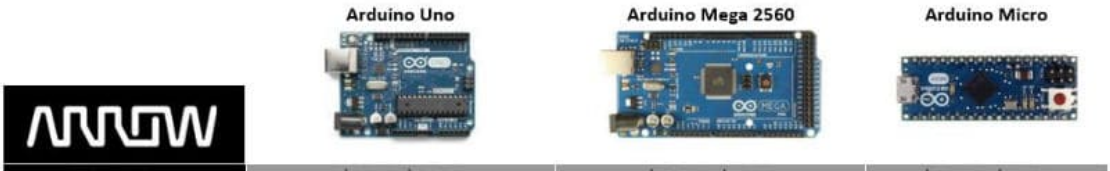
This section covers the design process for the Hardware-based subsystem associated to the VMI to be created as output of this project. The information herein will be structured as follows:

1. **Selection of an Arduino-compatible board**: this excerpt will discuss on the board upon which the whole hardware will be based, listing the arguments for its use.
2. **Selection of sensors and circuitry design**: this section will detail the sensors used and the reasons that meant its selection.
3. **Control algorithm design**: will depict the software algorithm to be run non-stop during the operation of the MIDI controller.

4.2.1.2.1 Selection of an Arduino-compatible board

Due to the hands-on experience with Arduino development the author of this dissertation had prior to the development of this project, an Arduino compatible board was decided as device upon which the final solution could be built. Since the task to be performed is fairly simple, a simple microcontroller board is known to suffice, so no device including an operating system was considered for the design. Several aspects were evaluated to determine the best option (to the eyes of the developer) among the existing boards.

To begin with, we may assume that all four limbs shall be used by the user to interact with the subsystem (the gesture controller), so that means that at least 4 sensors that capture interaction will be required to gather input. This interaction shall be in the shape of hits or pressure, resembling how the user interacts with a real drumset.Therefore, a must-have requirement for the final board was a minimum of 4 analog input pins. Luckily, most boards satisfy this requirement and thus, other aspects shall be considered. Amongst the most popular boards we can find ***Arduino Micro***, ***Arduino Uno*** and ***Arduino Mega 2560***. The differences between these three boards can be summarised in table 4.2:
From this table we can observe that all of the aforementioned boards provide similar characteristics, differentiating mostly in size and memory. Since the system is known to progressively grow in complexity, source code size, sensors and actuators attached to the board, flexibility was considered as a crucial decision factor. Whereas *Uno* and *Mega* were compatible with shields, something Arduino *Micro* did not allow, as the pins are in the form of through-holes, in contrast to the other two. This is something to be considered when trying to expand on the features provided by the gestural controller; Bluetooth communication or Wifi features could possibly be implemented in the future. This was the main reason for discarding *Micro*.
On the other hand, as more sensors were planned to be introduced in future iter-



	Arduino Uno	Arduino Mega 2560	Arduino Micro
Price Points	$19.99-$23.00	$36.61 - $39.00	$19.80 - $24.38
Dimension	2.7 in x 2.1 in	4 in x 2.1 in	0.7 in x 1.9 in
Processor	Atmega328P	ATmega2560	ATmega32U4
Clock Speed	16MHz	16MHz	16MHz
Flash Memory (kB)	32	256	32
EEPROM (kB)	1	4	1
SRAM (kB)	2	8	2.5
Voltage Level	5V	5V	5V
Digital I/O Pins	14	54	20
Digital I/O with PWM Pins	6	15	7
Analog Pins	6	16	12
USB Connectivity	Standard A/B USB	Standard A/B USB	Micro-USB
Shield Compatibility	Yes	Yes	No
Ethernet/Wi-FI/Bluetooth	No (a Shield/module can enable it)	No (a Shield/module can enable it)	No

Figure 4.2: Arduino Boards Comparison from [73]

ations, it was a nice feature to have support for SPI[7] and TWI[8] protocols, that could be used for communication between, say MPU9250[9] which could be used to obtain orientation information from the hands of the user or tangible interface used in further iterations on the project. All of the boards included support for these protocols so other aspects had to be evaluated to take a decision.
After a thorough evaluation, ***Arduino Mega*** was chosen. The advantages of this model over *Uno* are multiple:

- It comes with 4 UARTS, that is, many Serial ports can be connected to the Arduino, and those UARTs are in charge of transforming parallel information (more than one bit transmitted at a time) into a single line stream of bits. This is a great advantage over *Uno*, for example, since 0(RX) and 1(TX) pins are connected to the USB to TLL Serial chip and precisely that prevents you from using those pins at the same time as you are programmming the chip, because otherwise, conflicts happen. Since it is important to search for efficiency on debugging, the *Arduino Mega* was known to avoid such conflicts, as the rest of UART could be used for, say Bluetooth communication without generating issues when uploading software to the board.

- Its clock speed is 16MHz, which means that the oscillator within the chip generates 16,000,000 variations of voltage (oscillations) per second, which maps to at least one low-level operation being performed; ideally more. So more than 16,000,000 operations are performed per second, at least one operation every 625 μ s. This means that as each UART transmits one bit on every oscillation of the clock, a total amount of 10 bits (the size of a data byte plus auxiliary bits of the Serial protocol) take to be sent 6,250 μ s (0.00625ms), ideally providing

[7]Serial Peripheral Inteface, a communication protocol used to transmit data between integrated circuits that uses 4 buses

[8]Often referred to as I^2C, a protocol to communicate among Integrated Circuits that uses only 2 cables

[9]a 9DOF IMU

real-time output for the purpose of musical performance of drums (considering a maximum allowed delay of around 3ms and knowing that a the average size of a MIDI message is 3 data-bytes; the time it takes for a complete MIDI message to be sent is 0.01875 ms if there is no buffering). This feature was common to all the evaluated boards but equally important.

- *Mega* includes 14 PWM[10] pins, which allow to generate analog signals using digital means, allowing for a varying output from HIGH to LOW values which translate into a given portion of time (called duty cycle) connected to a 5V source. This on and off pattern allows to simulate an intermediate voltage. This one was the board which had the highest number of pins of this kind, so the feature was taken into account for the final decision.

- *Arduino Mega* has 6 available external interrupt PINS, from INT0 to INT5 in pins 2,3,21,20,19 and 18 respectively. This mechanisms allows for handling up to 6 inputs without polling (without constantly checking the value of the pin), which is desirable for expanding the capabilities of the system; in other words, it allows for multitasking and actually guaranteeing that no user input is lost. This was an important reason to pick *Arduino Mega* out from the set of discussed boards.

- Memory was an important aspect of the system, as the system is to be expanded over time and code size may eventually become an issue. *Arduino Mega* led the competition in this aspect, since it is equipped with a 256KB RAM, 8KB of SRAM and 4KB of EEPROM, which is more than double of the capabilities offered by other boards.

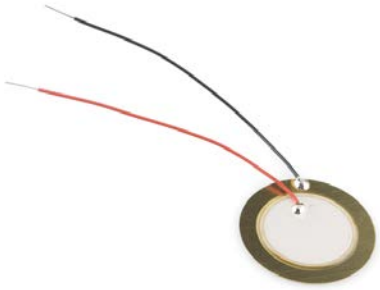
As an embedded system, the Hardware MIDI controller designed for this project was to be developed by means of a set of linearly executed tasks, running one after the other, controlling inputs and reacting to them appropriately. Thus, the MIDI Controller subsystem can be essentially thought as a **control system**, an open loop control system whose goal is to trigger a specific behaviour (sound) in the receiving end; that is, output from the control system is generated solely based on inputs.
The subsystem at hand is not only a control system but was also targeted at an embedded system, one with limitations regarding resources (the *Arduino Mega* board). Furthermore, any music controller can be categorised as well as a real-time system, a soft real-time system to be more precise, since there are timing requirements but no catastrophic disasters shall be produced due to the malfunctioning of this system, as it is not dealing with people's lives. However, incorrectly detecting a note can be disastrous for a performance, so we are seeking here for a level of precision that is able to provide playfulness to most musicians (drummers in this case). Details on the design of the actual solution can be found next.

[10]Pulse Wave Modulation

4.2.1.2.2 Sensor selection and circuitry design

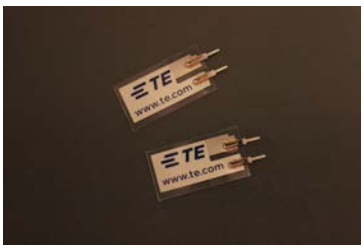
Starting from the assumption that interaction from the user has to be captured by means of hitting, and driven by the fact that the system shall somehow handle a pedal-like mechanism to deal with aperture of the hi-hat cymbal, we can safely state that at least 4 sensors are required, so that all user limbs can be leveraged simultaneously as input generators.
Due to the fact that stamping is a widely used way to practice rhythmic patterns, this first version of the subsystem was thought to include a sensor to detect that kind of interaction mode.
Accelerometers were evaluated but discarded among other digital sensors, which did not allow for velocity extraction straight away, so finally a **piezo sensor** was chosen as mechanism to detect the stamping intensity.
Regarding the hi-hat pedal counterpart, several alternatives were evaluated, including IMUs, piezos and **light sensors**. The latter turned out to be the simplest solution to the problem and proved to work well if carefully calibrated according to the initial light intensity sensed in the environment. Lastly, regarding hand interaction, **film piezos** were chosen for gathering inputs in an analog fashion.
To sum up, the Hardware-based subsystem is composed of **four sensors**, each one associated with one of the limbs:

- **Piezo sensor**: It serves as a sensor that simulates bass drum pedal hits, often triggered by the right foot. A piezoelectric disk generates a voltage when deformed, so it can be directly pressed or hidden in a practice pad for physically protecting itself and allowing the drummer to use an actual pedal for interaction.



- **Light sensor**: in order to control the opening of the hi-hat, this sensor, which words based on the light intensity it receives, simulates the interaction with a hi-hat pedal, since depending on the occlusion performed over the sensor (often with the left foot) we enable certain sounds on the synthesizer end.

- 2 **Analog hit sensors**: they allow for detecting vibrations that can be translated into MIDI velocity when appropriately mapped. Each of these sensors shall eventually be directly attached to a drumstick, in order to capture user input. These sensors were chosen to be flexible, so that they could later be attached to some kind of tangible interface.



The schematic of the system is shown in Figure 4.3. Two leds are used for providing the user with visual feedback regarding the state of the system, as required in the system requirements. The green led informs about how "closed" the hi-hat is, meaning that it is to be light up when the light sensor tends to be occluded, whereas the red LED tells whether the MIDI controller is powered up or not, which aims to provide a very basic intuition of what is happening within the subsystem.

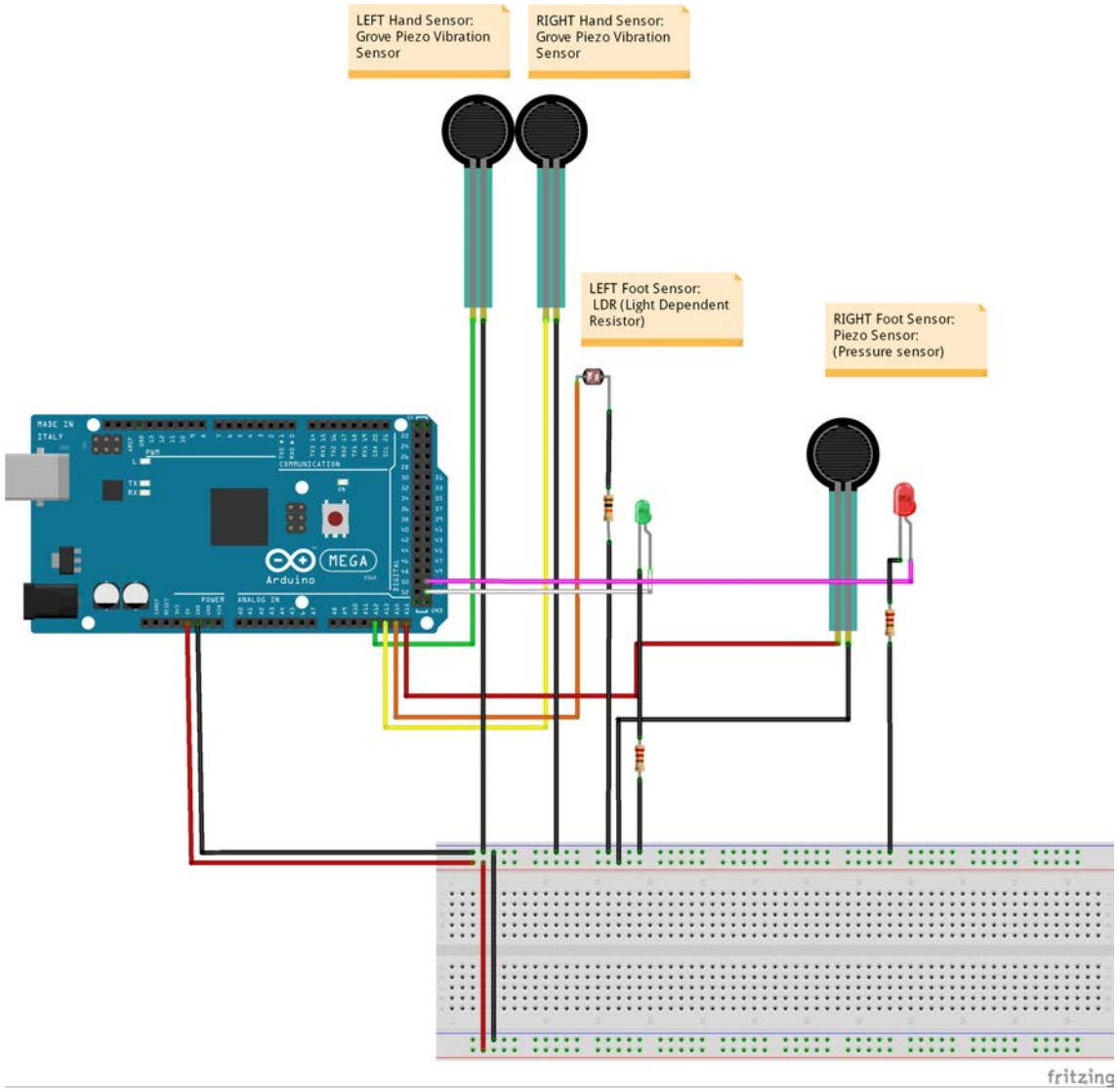


Figure 4.3: Arduino-Based MIDI Controller Schematic v1.0 (Bare Bones Build)

4.2.1.2.3 Design of the control algorithm

This section details the processing to be done of the inputs the user provides, briefly summarises how monitoring is performed and how MIDI output is generated.
To begin with, it is not possible to plot the gesture control problem as a cyclic scheduler due to the arbitrary timing of the user's input, which cannot be predicted in advance. Neither is it following a repeating pattern, so the program running within the micro-controller shall monitor the user's input, say "frequently enough" for most performances to be accurately detected and transmitted via USB, at least that is the level of accuracy we are seeking in this first iteration.
The proposed basic solution implies no timing guarantees, since hits may not be recognised if the user (a drummer) is able to play fast enough for different pin status reads (via Arduino *analogRead()*) not to detect sequential alterations in the input that occur within a very small time period (even though assumptions are made so that this is impossible for a human player) Anyways, the subsystem is functional due to the fact that inputs are verified at a huge rate, giving the illusion of them being processed in parallel.
The control algorithm to be implemented in the embedded system is the following:

For each sensor, there is a threshold, say *thresholdInteraction* (to be adjusted depending on the user or environmental conditions prior to interaction). Any value on the voltage read from the sensors surpassing such *thresholdInteraction* is interpreted as interaction, that is, as a hit to a given drum-set piece.
Since a single message per hit is to be sent but the sensors used are *analog*, that is to say, providing continuous values, the idea is to avoid duplication of *NoteOn* messages by assuming certain hits can't be feasibly performed in a very small period of time. We can assume a extreme case scenario to compute some values to use as reference, say we have a tempo of 220bpm, a fast tempo.
Imagine we want to play 16th notes at that beat, which means 3,67 hits per second with only one of the limbs (a huge speed indeed). Therefore, we could assume that two hits from the user will never happen within such time span, being this assumption pretty restrictive already.
In order to avoid processing triggered notes in the input "too often" (avoiding mistakenly identifying hits due to noise in the input), we keep two timers per input sensor. Thus, even though we continuously monitor the inputs, we only process them when a certain time interval is overflown. This interval is given by the aforementioned calculation, thus, set at 272.5 ms approximately
In the pseudo-code shown in Algorithm 1 the variable *timeCurr* keeps track of the most recent time "interaction" has occurred, meaning that the analog value associated with a specific limb has surpassed the threshold (for whatever shall be considered as a hit), to be set at the very beginning of the program.
On the other hand, *timeLast* keeps track of the timestamp at which the last processed "note" was identified.
Both are used to compute the time elapsed since a NoteOn message was triggered by a given limb, which is used to avoid accidentally sending two NoteOn messages that are in fact associated with a single hit from the user viewpoint. Note that interaction thresholds are to be decided based upon user preferences, since depending

on the style of music or playing style itself, the movements and force exerted when a hit is performed may vary. Customisation is made through direct code modification at this point, but different approaches may be leveraged in subsequent iterations over the project.

Algorithm 1 Input detection algorithm

Require:

- $thresholdInteraction_i$ is defined for each sensor independently, to adapt to player interaction mode (how much force or pressure is considered as interaction)
- $NoteFeasibleIntervalSensor_i$ represents a time interval within which we assume that two notes can't be "humanly" played using a single limb. This values are used to avoid processing user input incorrectly.

```
1:  while true do
2:      i ← 0
3:      while i ≤ numSensors do
4:          currValueSensor_i ← ReadSensor(i)
5:          if currValueSensor_i ≥ thresholdInteraction_i then
6:              timeCurrSensor_i ← getMilisecondsFromStartup()
7:          if currValueSensor_i ≥ thresholdInteraction_i
    and prevValueSensor_i ≤ thresholdInteraction_i
    and getElapsedTime(timeCurrSensor_i, timeLastSensor_i) >
    NoteFeasibleIntervalSensor_i then
8:              sendNoteOn(channel, getNoteMapping_sensor(i), velocity)
9:              timeLastSensor_i ← timeCurrSensor_i
10:         currValueSensor_i ← lastValueSensor_i
```

The algorithm shows a simple triggering of MIDI NoteOn messages when interaction is detected on a specific sensor, which has a specific MIDI number associated (called $getNoteMapping_{sensor}(i)$ in the code) . This numbers correspond to the most basic components of a drumset: the hi-hat, the snare drum and the bass drum. From this algorithm, we can observe that the approach to sound generation is fixed for this specific build; i.e. each sensor has only one sound associated (except for the LDR).In addition, as the reader may have observed, no NoteOff messages are used in the algorithm or previous explanation whatsoever. This is due to the nature of non-pitched percussion sounds, whose duration is limited, so it is not strictly necessary to include NoteOff messages to shut sounds down, as it would be the case for a piano keyboard. This decision is accurate for sounds which are characterised by short attack and release times but not for cymbals, whose sound is often sustained for longer periods of time. NoteOff messages would be useful to implement cymbal choking, which is an interaction form that goes beyond the scope of this dissertation and may be explored in future work.

4.2.1.2.4 Hi-hat interaction handling

As mentioned earlier, one of the sensors' input (LDR's) is handled differently since is targeted at resembling the way a hi-hat pedal works, and therefore this one will work differently in the sense that the sensor associated with the right hand drumstick (at

some point of the development) will be mapped to different sounds in the receiving end, simulating the real interaction mode that allows regulating the aperture of the hi-hat by the fact of detecting that more or less amount light is getting to the light sensor.
This is strictly necessary for this build due to requirements on the number of sounds that the system as a whole shall be able to generate, which shall exceed three, and which is impossible to achieve unless different sounds on the simulated hi-hat can be generated.
As a rather simple design, we chose for this particular build a binary status allowed for the pedal. That means that from the point of view of the code, the pedal can only be either OPEN or CLOSED, and that means two different sounds shall be triggered in the sound generator subsystem depending on the state of the hi-hat.
In order to achieve this, we keep track of the analog status of the LDR and set a threshold that determines the logic to decide whether the hi-hat pedal is open or closed, a value that consequently alters the noteNumber included in the MIDI NoteOn messages sent as output that are associated to the right-hand interaction of the user.

Two timers are kept to avoid processing of hits "too often", as it was shown in Algorithm 1.

Specific parameters and sizes of data structures are implementation dependent and therefore not specified here, you can find more about them in the code itself, where the detailed design is embodied for this project. All implementation decisions were made looking for a trade-off between precision and performance loses, ultimately seeking for usability of the product as a whole as musically-useful.

Algorithm 2 Hi hat handling algorithm Bare Bones Build

Require:

- *thresholdHHClosed* is set depending on the environment's light; values below this threshold imply a closed hi-hat noteNumber to be generated as output.
- *thresholdInteraction* is set to the minimum value as input that shall be considered a virtual hit.
- *noteFeasibleIntervalSensorHH* represents a time interval within which we assume that two notes can't be "humanly" played using a single limb, two hits on the hi-hat can't be feasibly performed within this timespan.
- *hiHatStatus* is initially set to CLOSED or OPEN and updated according to the read input on the LDR

```
1:  while true do
2:      currValueHH← readHHSensor()
3:      currValueHitSensorHH ← readHitSensorHH()
4:      if currValueHitSensorHH ≥ thresholdInteraction then
5:          timeCurrHHSensor ← getMilisecondsFromStartup()
6:      if
7:          currValueHH ≥ thresholdHHClosed and
8:          prevValueHH ≤ thresholdHHClosed and
9:          getElapsedTime(timeCurrHHSensor, timeLastHHSensor)
    > NoteFeasibleIntervalSensor then
10:         midiVelocity ← computeVelocity(currValueHH)
11:         if hiHatStatus is CLOSED then
12:             sendNoteOn(channel, hhMappingClosed, midiVelocity)
13:         else
14:             sendNoteOn(channel, hhMappingOpen, midiVelocity)
15:         timeLastHHSensor ← timeCurrHHSensor
16:     currValueHHSensor_i ← lastValueHHSensor
```

4.2.1.3 Software: Unity-based Sound generator

Before diving into design of the Bare Bones solution to the problem, we may justify the selection of a set of technologies as building blocks for the system to come to life, whose selection is derived from the software requirements.
Firstly, the software subsystem was developed using **Unity**. This decision was taken owing to several reasons:

- It was an **opportunity to learn about videogame creation**, a topic of great interest and demand in the IT sector. The idea of getting started with tool by means of this project could be consequently positive in two aspects: personal enrichment and potential of showing a small expertise with the tool to employer; or as a self-assessment of knowledge-base for possible further studies.
- Since visualisation is the main purpose (in conjunction with sound generation) of the Sound generating subsystem, game engines are known **to ease and speed up the development process** of 'game alike programs', and are designed to provide facilities with a focus on dealing with 3D models, 3D audio and User Interfaces. These are the main items to be dealt with in the design and implementation of the subsystem so it seems like the best approach to follow under the limited time-span planned for completion of the system.
- **Unity was chosen** over other game engines because of the quality documentation and popularity. Needless to say that it is free, in contrast with other game engines with similar features and workflows [74]. And it supports Windows 10 as target, allowing us to complete SR-NF-21.

In addition, regarding the approach to handling MIDI input, we based the design of this initial build on the **MIDI Jack** Unity add-on, which keeps track of the state of all the different 16 MIDI channels, notes that are played, when they have been played and so on. Next, a thorough description of the MIDI Jack add-on and its capabilities is provided, as it may help readers understand how the Unity programming paradigm works and why the solution proposed later makes sense.

4.2.1.3.1 About MIDI Jack:

The Plugin comprises a set of 5 C# files, some of which make use of a DLL whose source code is available at keijiro's github page [72]. Each file has its own purpose, summarised below:

- **MidiJackWindow.cs** is a file implementing a GUI window for the Unity editor, which shows the data messages being received through all MIDI ports in the system; so basically serves the purpose of debugging, allowing for the messages being received to be displayed after they are processed, accessing them from a message history data structure. This file is not interesting for the

design part because it doesn't provide any useful functionality to be leveraged in the project; at least for the implementation as such.

- **Midi.cs** defines the data structures that stores data associated to a given MIDI message, either CC or Noteon/NoteOff. Encapsulates the processing of the input data stream and separates the different logical fields of it for an easy access and logical reasoning in the code. Thus, this file provides with the innermost data structure (the MIDI message and its fields), used and dealt with later on higher abstraction levels.

- **MidiDriver.cs** implements the main logic of the add-on and provides with internal structures that store the status of MIDI notes and knobs (the *ChannelState* class), public methods to access the aforementioned structures as well as the core function providing barely all capabilities, the inner Update() method. This method does the following.

    1. Checks the status of all the data structures holding MIDI in data to guarantee that the values are up to date. Since Unity works mostly based on frames for visualization purposes and that is one of the typical approaches to gameplay programming, the addon provides functions that allow other programmers to know whether data was received or triggered during the current frame, released or simply ON or OFF at any given point in time.
    2. Processes the incoming MIDI messages, whose low level details are left to DLL functions; updating the data structures associated with each MIDI Channel to reflect the state of the MIDI input.
    3. Notifies about events defined with C# delegates, allowing classes subscribed to the events to execute methods in their scope. This is a useful tool to allow users of this plugin to create low coupled programs that respond to certain events, such as the receipt of a NoteOn message, for example; upon which the implementation of the virtual drumset is based.

- **MidiStateUpdater.cs** is a file which derives from MonoBehavior and therefore, the one attached to a Unity GameObject. It is the one which gets to execute the core functionality of the Addon, once its creation is performed by the constructor of the MidiDriver class. This class uses a delegate as a way to receive a function reference as parameter, using it to determine what code to execute within the Update() method defined within.

- **MidiMaster.cs** is a static class that provides a high-level access to all the main features of the add-on, allowing directly calling methods without requiring an instance of any other MIDIJack class. It is the one to be referenced within our code directly, as it acts as a clear API for programmers aiming at rapidly incorporating MIDI input support to in Unity.

In order to support the explanation and aiming to straightforwardly understand the complex flow of the add-on classes (how they relate to each other), Figure 4.4 is provided.

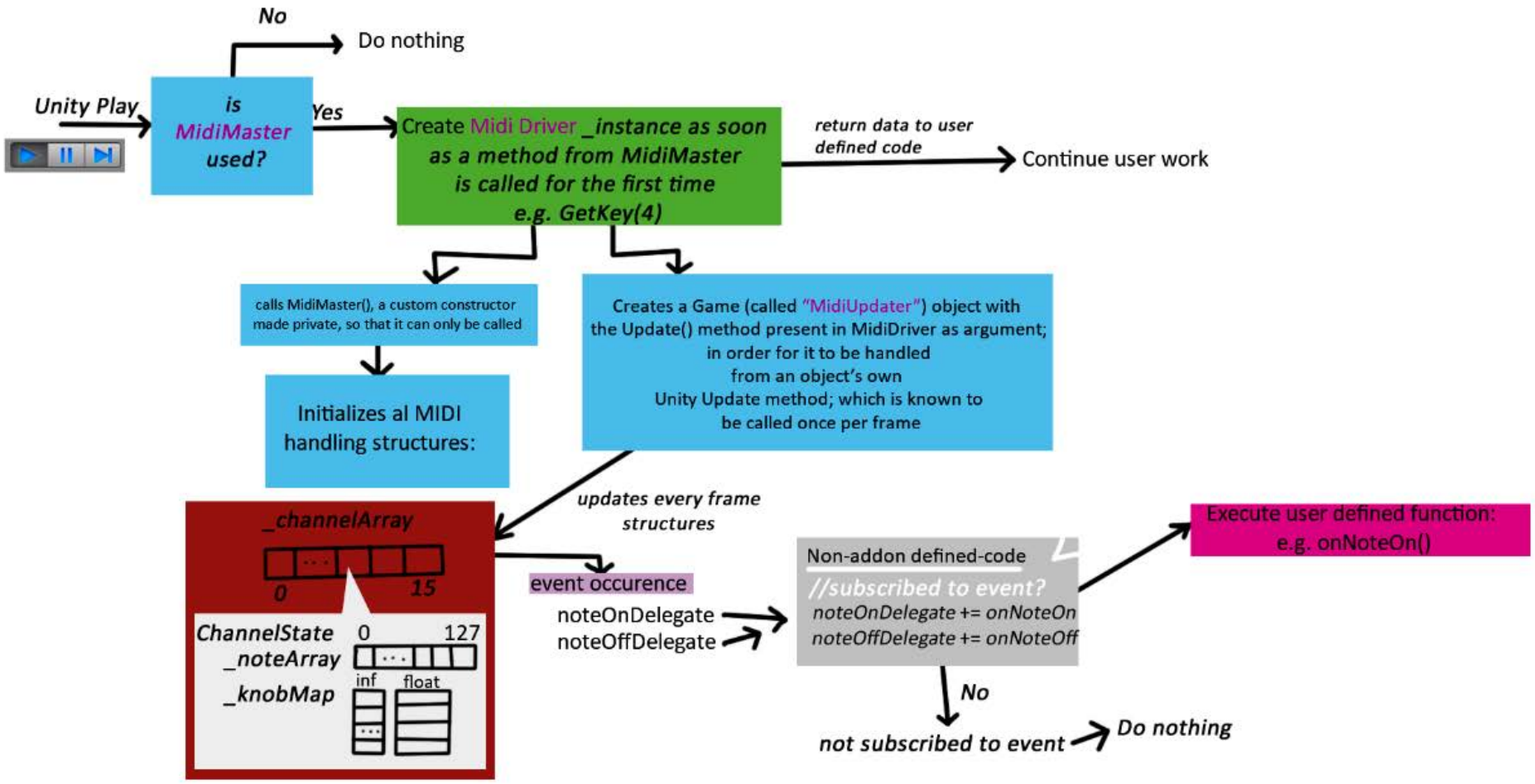


Figure 4.4: MIDI Jack execution logic diagram

4.2.1.3.2 Architectural design of the videogame-alike program

In this excerpt, the Architectural design will be formally depicted using the OOD Methodology (Object Oriented Design). This one was selected because of the nature of the videogame-creation environment (Unity), which allows to create code using C# , an OOP(Object Oriented Programming) language, meaning that its logic is based on classes and objects that are the virtual representations of entities (logical or physical) that exist in the real-world.

The following **components** were identified in the system, each one targeted at implementing a clear overall purposeful function. All of the mainly involved components are listed (both custom and pre-existing), in order to guide the understanding from the reader.

- **MIDIJack** (C1): the add-on acts as a component encapsulating MIDI input and allowing other components to access related data by means of a clearly defined interface. No substantial changes need to be performed in this pre-existent component. It allows compliance with SR-NF-07(see 3.64) and SR-FR-16(see 3.51).
- **Drumset Mapper** (C2): in charge of mapping MIDI Input numbers to actions using the event notification tools provided by MIDI Jack and is responsible for configuring the sounds that are to be played in response to user input and animations to be executed. This is a brand new component.
- **Drumsetpiece Handler** (C3): handles the information related to specific parts of a virtual drumset and is in charge of generating changes on the visualised objects as well as directly triggering 3D audio. This is a brand new component.
- **Unity Audio** (C4): classes dealing with Audio in Unity are used by custom components to handle sound generation, clip assignment and sound configuration. No changes will be performed to this component.
- **Unity Renderer** (C5): the classes handling materials, object's look and textures in Unity will be used by custom components in order to trigger animations. No changes will be performed to this component either.

Figure 4.5 shows custom components in yellow colour, while pre-existent components are shown in a bluish green colour.
A detailed specification of each custom components is provided using the following table template; containing the following fields:

- **ID**: follows the pattern CX where X increases every new component, so that each component can be univocally identified.
- **Origin**: defines whether the Component was created for the sole purpose of this dissertation (Custom) or if it existed previous to it (Pre-existent).

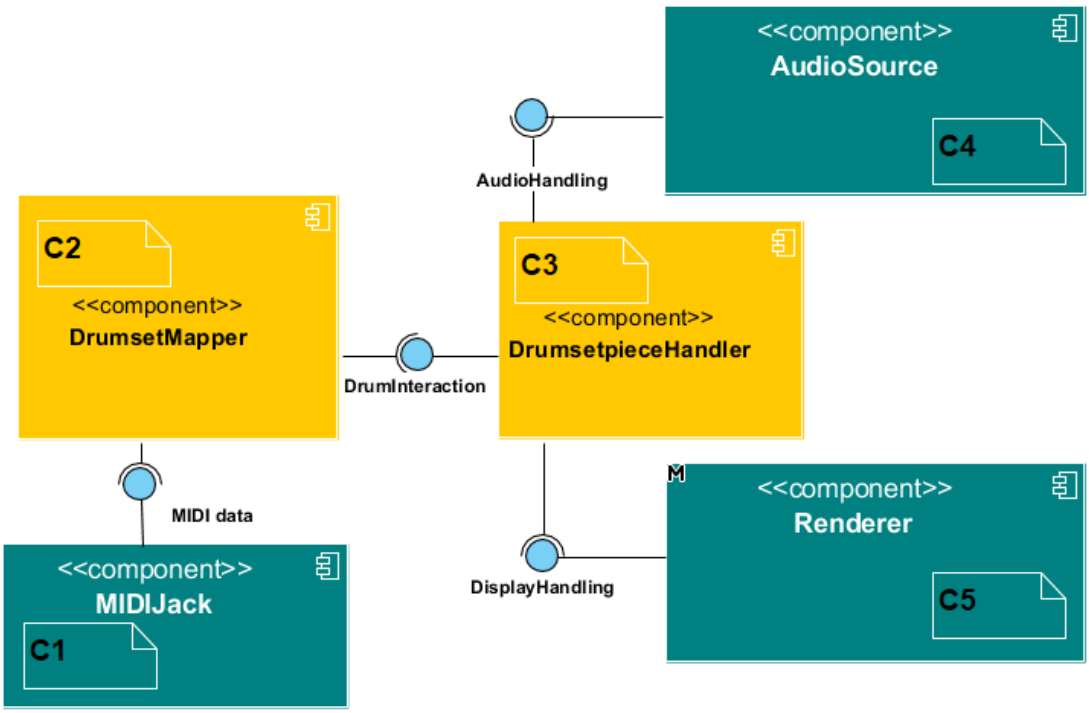


Figure 4.5: (Sound generator) Component Diagram for Bare Bones Build

- **Purpose**: references the Software requirements related to the component.
- **Function**: states what the component does; that is, the process it encapsulates as well as the information stored or transmitted.
- **Type**: can be Executable (transforms data) or Non-Executable (does not directly transform data, only does data management).
- **Dependencies**: they are represented by the hooks attached to lollipops in the Component Diagram . They are used interfaces.
- **Offered interfaces**: refer to the interfaces each specific component offers, represented as lollipops or circles coming out of a given component.
- **Target Build**: the associated Build towards which this design has been performed; there are two possible values; Modest Build or Bare Bones Build.

A Class diagram is provided in Figure 4.5 for the software system developed in this build. It aims to clarify how components may be implemented, identifying responsibilities, relationships and purpose of each of the classes and components involved in the solution.

Component Specification Template	
ID	e.g. Made-up Component (C1)
Origin	Custom | Pre-existent
Purpose	e.g. SR-FR-01, SR-NF-01 ...
Function	Brief description of the responsibility of the component within the system.
Type	Executable | Non-executable
Dependencies	e.g. **InterfaceIn**: the component uses it to access the information from the database, whose low-level details are encapsulated in the component offering such interface.
Offered interfaces	e.g. **InterfaceOut**: set of functionalities offered to the public that enable access to status of the user within the application lifecycle.
Target build	Bare Bones Build | Modest Build

Table 4.1: Component Specification Template

Component Specification	
ID	DrumsetMapper(C2)
Origin	Custom
Purpose	**SR-FR-02**,**SR-FR-08**,**SR-FR-09**,**SR-NF-19** and**SR-FR-03**
Function	0.Subscribes to the noteOn event from C1. 1.Assigns Midi Numbers as input to Drumpieces; forwarding input to C3. 2.Loads sounds to be triggered from library of samples. 3.Assigns sounds and configures its parameters (looping, playOnAwake...)
Type	Executable
Dependencies	**MIDIJack's (C1) Interface MIDI Data**: obtains the information regarding the state of the MIDI input, allowing for easy logical access to notes that are on, off, status of the knobs...
Offered interfaces	None
Target build	Bare Bones Build

Table 4.2: Component 2 Specification Bare Bones Build

Component Specification	
ID	Drumsepiece handler (C3)
Origin	Custom
Purpose	**SR-FR-01**,**SR-FR-07**,**SR-NF-06**,**SR-FR-10**,**SR-NF-08**,**SR-NF-19** and**SR-NF-20**
Function	0.Encapsulates all the data associated with the virtual representation of a drumkit piece. 1.Enables sound triggering. 2.Enables material animation.
Type	Non-Executable
Dependencies	**Unity's Audio (C4) Interface Audio Handling**: provides functionalities regarding audio playback, 3d audio spatialization, audio playback status access, etc. **Unity's Renderer (C5) Interface DisplayHandling**: allows changes of material assigned to 3D objects in the scene.
Offered interfaces	**Interface DrumInteraction**:enables modification of Materials associated to a specific 3D object (a virtual drumset piece; using C5 utilities) and triggers 3d sounds using other components (C4).
Target build	Bare Bones Build

Table 4.3: Component 3 Specification Bare Bones Build

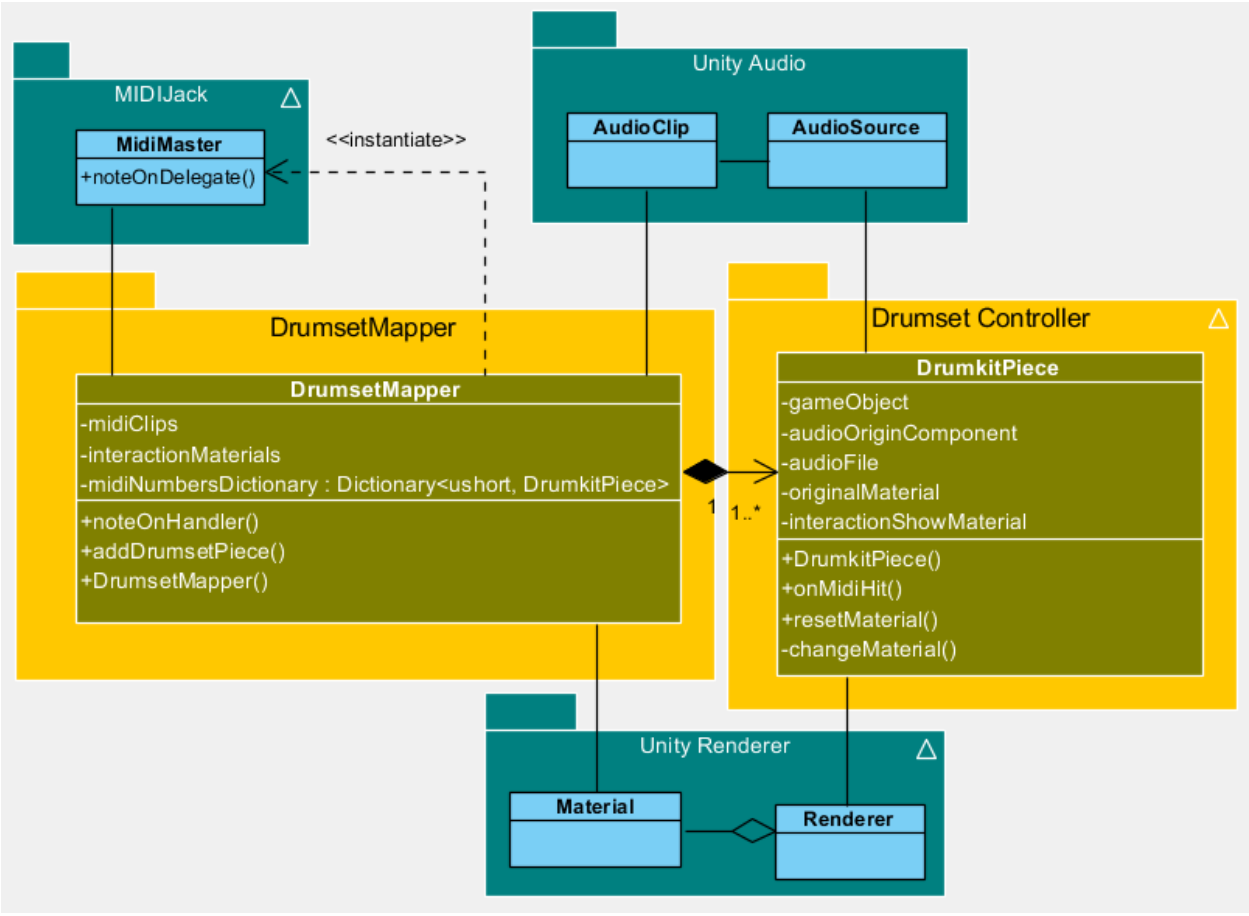


Figure 4.6: (Sound generator) Class Diagram for Bare Bones Build

In short, the program workflow would work as follows:

1. The **Drumset Mapper** would get assigned all the *midiClips* that the user will be able to trigger during program execution, as well as a material that would substitute the one present in any drumset piece with which the user may interact in the future.

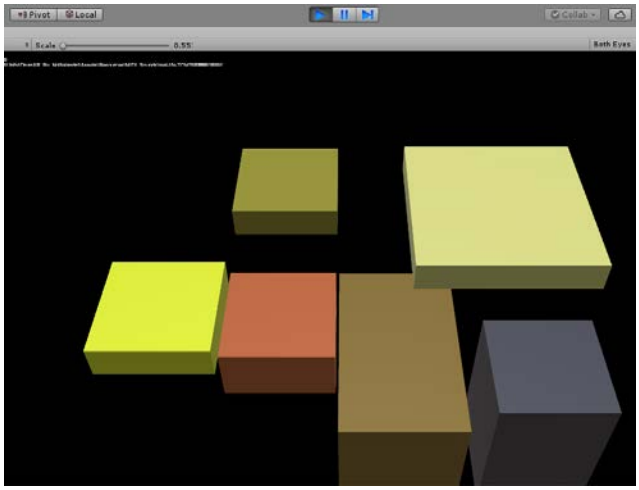
2. To provide with mappings between MIDI inputs and sounds/animation, a Dictionary, say *midiNumbersDictionary* would keep the relation among the noteNumber received and the virtual drum piece to be conceptually "hit". The mapper would be in charge to create instances of *DrumkitPiece* (See Drumset Controller Component) to represent and store data related to a virtual drum piece, allowing for later sound triggering and animation.

3. Using the *noteOnDelegate* delegate offered by MIDI Jack, made public via *MidiMaster.cs*, we would create a function that processes the received notes for all of the drum pieces that have an assignment. On initialisation of the program, that is to say Unity's ***Awake()***, we subscribe to that event; meaning that we add the created method to the list of functions to be called on event execution (i.e when a note that wants to be handled is received via any MIDI port). In this way, the custom implementation is only in charge of extracting the noteNumber associated to a given noteOn message received; and process it by looking for the key (the noteNumber) within the *midiNumbersDictionary* dictionary to finally use the *DrumkitPiece class* methods associated to the extracted object to trigger the desired sound sample and alter the visuals of the desired drumset piece.

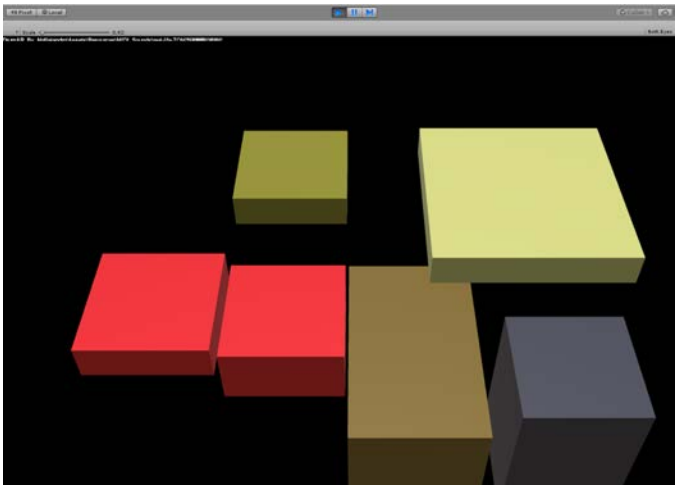
4.2.1.3.3 User Interface and visualization

For the visuals of the sound-generating subsystem, a very simplified drumset was created (see Figure 4.7a), made of a set of rectangular prisms with different materials, each one associated to a drum or cymbal within the simulated drumset. This simplification aimed to speed up the development process, leaving the fidelity of the graphical representation of the drumset for further iterations over the system. The figure shows both the visuals of the sound generation subsystem at rest and when it receives MIDI messages associated to, in this specific case, the hi-hat cymbal and the snare drum, which turn red on the very next frame when input associated to their drumpieces per se is received.

Midi mappings are strictly fixed and simplified for this version, not being compliant with MIDI standard assignments. These configuration parameters were made available through the Unity inspector to allow easy configuration without modifying code, which made testing much easier. Assignments can be found in Figure 4.8, which include noteNumbers in the interval [1-6].



(a) Bare Bones 3D visuals in Unity



(b) Bare Bones visuals on Interaction

Figure 4.7: Bare Bones Build 3D visuals

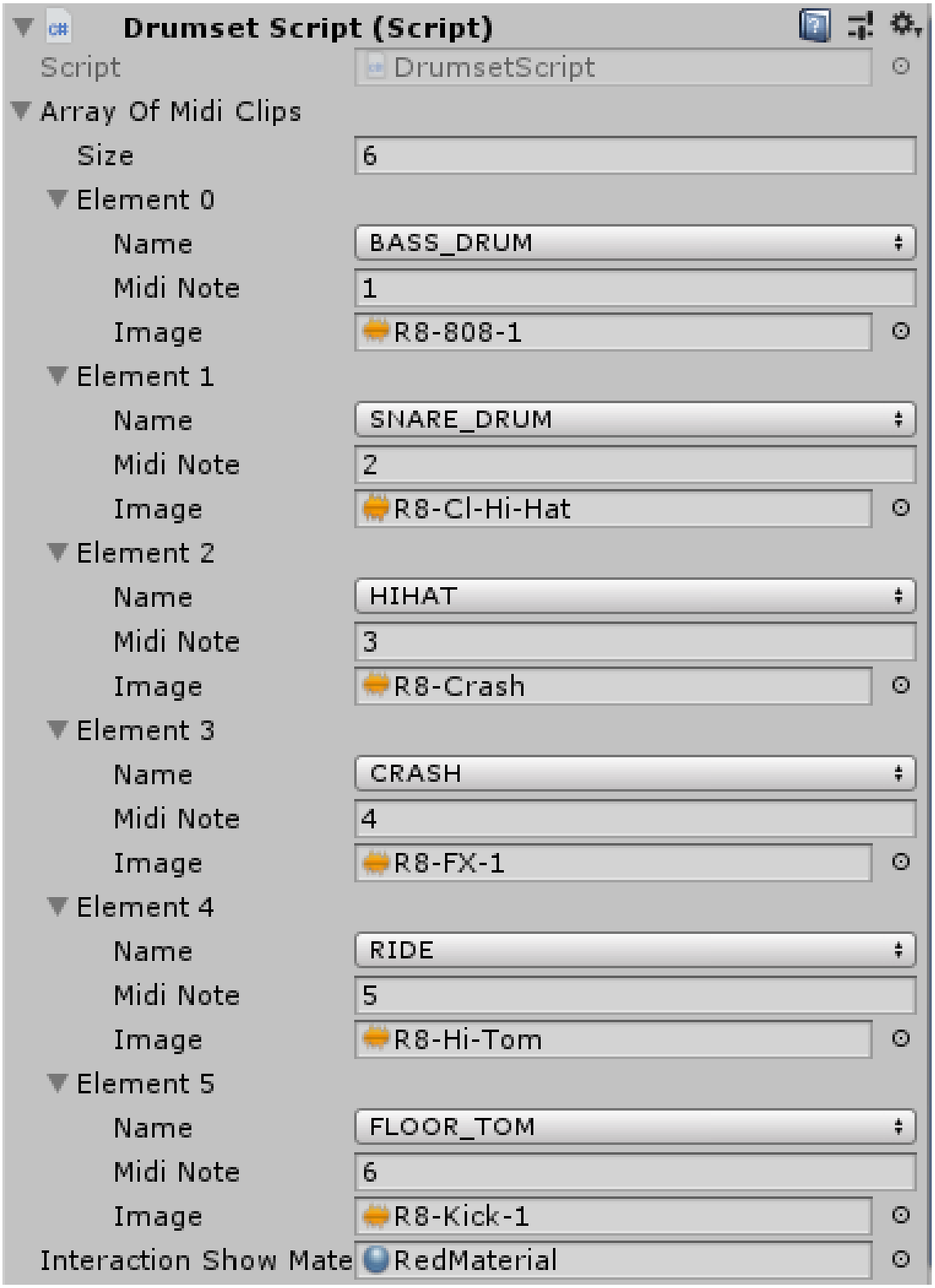


Figure 4.8: Midi Mappings in Bare Bones Build

4.2.2 Modest Build

As in the previous iteration of the project, the structure of the design will be analogously described next. In this build, called *Modest Build*, an improved version of the system is to be created, whose requirements are simplified to natural language in the first part of this passage (see 4.2.2.1). Later, gesture controller enhancements and software-side new features will be detailed in sections 4.2.2.2 and 4.2.2.3 respectively.

4.2.2.1 Modest Build Concept

The second build has as target to improve the whole system by making it more usable and customizable than the one obtained as outcome of the Bare Bones Build. Many objectives to this build were derived from reading several sets of guidelines for creating VRMIs and NIMEs (Jordà's [50], Perry Cook's [25] and [75], Ge Wang's [32], Serafin et al. [31]) and the author's own well-founded opinion after research and State-of-the-art reading; so you may find references to those later in this document. In fact, by simply iterating on this project, we are fulfilling Ge Wang's 11^{th} principle from [32].
Broadly speaking the following build is to achieve the following:

- **Improving the gesture controller**: based on the previously generated subsystem, the goal is to improve its usability, accuracy, expressiveness and portability by complying with the following main objectives:
  - **Providing time guarantees**: the system shall be able to be responsive enough to handle two hits happening within a maximum range of 68ms of separation (from the same sensor); that means we must measure performance to make sure no output cannot be sensed.
  - **Embedding sensors in a set of drumsticks and pedals**, in order to give birth to a tangible interface which can make the instrument more straightforward to use and allow the player to reuse their drumming skills.
  - **Implementing support for CC messages**: so that these can be used to control hi-hat aperture using continuous values, which is the implementation approach followed by most VSTs (such as Addictive Drums 2).
- **Improving the sound generator subsystem**: based on the simple solution obtained as output from the *Bare Bones* development, we aim to add new functionality that make of this virtual instrument a better fit for a musician and their creative minds, providing with custom mappings, volume settings and higher-fidelity 3D Visuals.
  - **Modelling an accurate representation of a Drumset in Blender**: this would provide with better graphics and immersion, since graphical and physical representations will be similar to each other, making the UI look, in addition, much more professional.

- **Expanding the number of sounds that can be generated by supporting CC input**: this feature will enable a much higher range of hi-hat sounds, depending on the aperture value at a given point in time.
- **Implementing a configuration menu that allows changing sound-to-noteNumber-to-drum piece** configuration, as well as volume values for the drumset as a whole.
- **Implementing a sound-preview functionality**; this would allow users to listen to a sample that is to be assigned to a specific drum piece before actually assigning it, so that users can make an efficient use of the configuration menu.
- **Implementing support for camera movement**: would allow the user to explore the virtual environment, as if they were able to move their gaze around it.

4.2.2.2 Hardware Design: Arduino-Based MIDI Controller

One of the problems discussed by Jordà regarding "splitting the chain" (having a controller-sound generator architecture), was precisely the fact that it reduces the feel of control over the instrument. In order to address this problem and so that we can make interaction with our sound generator as organic as possible, we introduced a set of improvements to the gesture controller obtained in the previous design iteration.

4.2.2.2.1 Tangible Interface Design

Firstly, it was decided to attach the sensors of the system to something physical, which could be swung or hit around, and therefore, could resemble the way drummers interact with an real-world drumset. This decision may be thought as derived from Ge Wang's 4^{th} principle, "Induce viewer to experience substance"; which means, one should keep the attention of the user away from the artefact that makes possible the music, instead, let they focus on playing. It also relates to the 9^{th} principle, "Be whimsical and organic"; in the sense that we involve real-world interaction paradigms: as it is hitting.
Figure 4.9 shows the assembly performed to embed sensors within a set of real world objects in order to make them more usable. Female USB connectors were soldered to both the Arduino inputs and the tangible interface, in order to ease playing and portability, since otherwise long thin cables would be necessary to interconnect sensors to the Arduino board and these would likely make connections fall apart due to the tightness associated to playing.

(a) Hi-hat pedal tangible interface

(b) Hand-held interface and playing setup

Figure 4.9: Tangible interface designed for ***DrumVR***

4.2.2.2.2 CC Hihat output design (improvement)

In contrast to the Bare Bones Build, which implemented hi-hat sound triggering as different noteNumbers sent depending on the two logical states of the hi-hat (OPEN or CLOSED) handled within the gesture controller, this build is to be stateless in this aspect. This means that conceptually, it will be the sound-generator, the one handling which sound shall be generated depending on the simulated hi-hat aperture, which will be continuously notified to that end.

In order for the receiving end (the sound generator) to be able to know how "closed" the simulated hi-hat is without providing with fixed note numbers within MIDI NoteOn output; MIDI CC messages are sent by the Arduino MIDI controller to let the Sound generating subsystem know about the state of the pedal more accurately, allowing for expansion of sounds that are mapped to the right hand interaction (hits to the simulated hi-hat).
Control Change dispatching is handled in a special way so that these messages do not overload the Serial bus with unnecessarily precise data, that is, we avoid providing the Sound generating subsystem with aperture data unless a substantial change has happened.

The exact approach to this problem consists in keeping an data structure with not only the exact previous value read from the light sensor, but instead keeping track of several of them. Using this set of previous values, we are able to determine more precisely the trend of the data being analysed by computing an average value. In this way, we can then compare the last detected analog value coming from the LDR (the hi-hat pedal counterpart) with the mean value calculated before to spot whether the pedal interaction is suffering a change, meaning that a substantial variation in the light getting to the light sensor is happening. This, consequently allows us to determine more accurately when the receiving end should be notified.
Once computed the mean value, we set a threshold of maximum deviation (*maxDeviation* in the algorithms) from the average value and use that as a boolean condition to be used as decision maker, since a current value that differs too much from the mean computed value logically means that a substantial change just happened.

Algorithm 3 shows how, by means of a function, the system is to conclude that a substantial change has happened in terms of light variation from the hi-hat pedal. A mean value is computed using a number n of samples taken over time regarding the LDR state. When the absolute value of the substraction of LDR current state and the mean exceeds the *maxDeviation* value, then, the function returns true, meaning that a change in the input is sufficient to notify the sound-generating subsystem.

Algorithm 4 represents how the CC output is to be handled inside the control loop, which is running all the time while the Arduino board is powered on. Note that *ccNumber* corresponds to number 4, which the standard value to use for "Foot Controller" as given by the MIDI Specification [11]. The value included within the

[11] CC Standard values: see https://www.midi.org/specifications-old/item/table-3-control-

Algorithm 3 Function to handle CC output overload

Require:

- $maxDeviation$ defined to distinguish 'substantial change' in LDR input
- n defining the number of samples to use in order to compute the mean
- $prevHihat$ an array of n positions which stores a set of sample values previous to the current one
- $currentHH$ is the value of the last sensed LDR pin value

1: **function** HIHATSTATEHASCHANGED
2: $sum \leftarrow 0$ # Variable to store $\sum_{t=0}^{n-1} prevHihat_i$
3: $mean \leftarrow 0$
4: **for** $i \leftarrow$0 **to** n-1 **do** { sum $\leftarrow sum + prevHiHat_i$}
5: $mean \leftarrow \frac{sum}{n}$ # Compute the mean value in the last n consecutive timestamps
6: **if** $|currentHH - mean| \geq maxDeviation$ **then**
7: **return** true
8: **return** false

CC MIDI output message is computed in Line 4 of the algorithm, which computes a percentage of aperture to later map into the [0-127] range allowed by the MIDI message specification.

4.2.2.3 Software: Unity-based Sound generator

4.2.2.3.1 Architectural Design

In this excerpt, the Architectural design will be once again formally depicted using the OOD Methodology (Object Oriented Design). A stress will be made in new components or redefined ones, briefly summarising the responsibilities each one has assigned. The following **components** were identified in the system, each one targeted at implementing a clear overall purposeful function; all of the mainly involved components are listed (both custom and pre-existing).

- **MIDIJack** (C1): As in the previous build, it offers utilities to get the status of MIDI input, delegates to execute custom-defined methods if they are subscribed and so on. It encapsulates all the MIDI in processing and makes it easily accessible to the Unity programmer.
- **Game Controller** (C2): Centralizes mapping data and controls main visuals of the videogame-like program, toggling the main menu and actually subscribing to C1 events to enable the main flow of the system. It replaces the Drumset Mapper (C2) from the previous version, and receives a different name and responsibilities that relate more closely to one another.

change-messages-data-bytes-2

- **Drumset controller** (C3): Drumset Controller (previously named DrumsetpieceHandler, C3) is an enhancement to the previous component, which identifies different types of drumset pieces such as generic ones and hi-hat, which in contrast to other pieces, generates more than one sound depending on the CC value received from the gesture controller. Inheritance is used to reduce code size and maintainability while providing special handling of specific drumkit pieces. These include new functions, among those, capabilities that allow to modify the sounds assigned to each 3D drum piece during runtime, something that is managed by the new component described later (C7).

- **Unity Audio** (C4): Audio playback, spatialization, volume and related utilities are accessible using this component, which enables sound generation and sound reassignment, as well as sound preview are sample library loading.

- **Unity Renderer** (C5): As described in the previous version, it enables hadling how 3D objects look within the virtual environment, allowing to change Materials during runtime.

- **Audio Mixer** (C6): Enables volume control of the whole drumset sound, which is a new customization functionality to be implemented in this build.

- **Configuration** (C7): This is a brand new component created for this build which handles everything that has to do with configuration or customization of the sound generator. It controls volume and menu look-and-feel,is in charge of loading library samples, creating sub-menus, assigning noteNumbers to Drum pieces and sounds by default, and enabling later alteration of those mappings.

Figure 4.10 shows components and interfaces among them. Red components are brand new components, added for the purpose of including new features required for this build; yellowish components are re-definitions and redesigns of programmer-defined components from the previous build, whereas blueish components are pre-existent; non programmer-defined ones.

Algorithm 4 Hi-Hat CC output and sampling management

Require:

- $prev_hh_c$ is the value [0-127] last sent into a MIDI CC message
- $activeChannel$ MIDI channel [0-15] towards which the MIDI CC message is sent
- $ccNumber$ the knob number to be used as hi-hat aperture handler, set to 4 (as in Standard MIDI has got the name "Foot Controller" assigned)
- $prevHihat$ an array of n positions which stores a set of sample values previous to the current one
- $currentHH$ is the value of the last sensed LDR pin value
- $sampleIt$ keeps track of the position in $prevHihat$ that is to be written next.
- n defining the number of samples to use in order to compute the mean
- $maxHH$ max value that the LDR sensed input can obtain, based on initial calibration

1: **while** true **do**
2: # Rest of the control loop work
3: **if** HIHATSTATEHASCHANGED **then**
4: $hh_cc \leftarrow currentHH * maxHH * 127$
5: **if** $prev_hh_cc \neq hh_cc$ **then**
6: SENDCONTROLCHANGEMESSAGE($activeChannel$,$ccNumber$,hh_cc)
Sends MIDI CC message over Serial
7: $sampleIt \leftarrow sampleIt \bmod n$
8: $prevHihat_{sampleIt} \leftarrow currentHH$
9: $sampleIt \leftarrow sampleIt + 1$

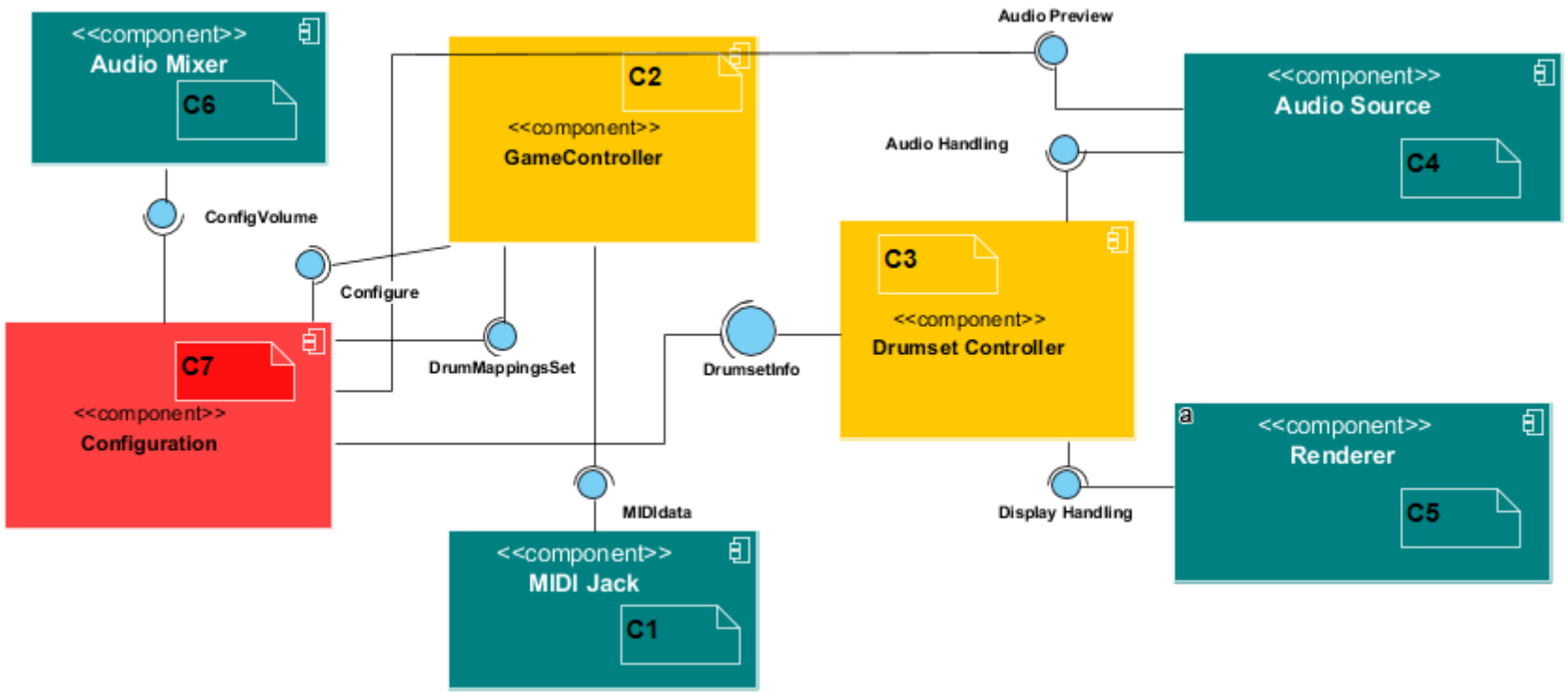


Figure 4.10: Component Diagram Modest Build

A detailed specification of each individual component is provided using the previously covered table template (see table 4.1); with the same fields that can be found in the previous build.

Component Specification Template	
ID	GameController(C2)
Origin	Custom
Purpose	**SR-FR-19** and **SR-NF-27**.
Function	0.Enables camera movement, changing user view. 1.Allows for rapid configuration of the CC knob to be mapped to hi-hat aperture. 2.subscribes to noteOnDelegate event 3.stores Mappings between midi noteNumbers and Drumpieces, data main store, accessible to all other components.
Type	Executable
Dependencies	**MIDIJack** (C1) **Interface MIDI Data:** obtains the information regarding the state of the MIDI input, allowing for easy logical access to notes that are on, off, status of the knobs, etc.
Offered interfaces	**Interface DrumMappingsSet**: allows access to the dictionary of mappings, of centralized use in this component.
Target build	Modest Build

Table 4.4: Component 2 Specification Modest Build

Component Specification	
ID	DrumsetController(C3)
Origin	Custom
Purpose	**SR-FR-18**
Function	0. Encapsulates all the data associated with the virtual representation of a drumkit piece. 1. Enables sound triggering. 2.Enables material animation. 3.Enables sound remapping and reassignment. 4.Enables multiple sounds to be generated depending on CC values.
Type	Non-Executable
Dependencies	**Audio Source** (C4) **Interface AudioHandling**: allows generation of audio with variable volume. **Renderer**(C5) **Interface DisplayHandling**: Enables animation of 3d objects representing drum pieces.
Offered interfaces	**Interface DrumsetInfo** : Allows mapping of Configuration Submenus to Drum Piece information and references.
Target build	Modest Build

Table 4.5: Component 3 Specification Modest Build

Component Specification	
ID	Configuration (C7)
Origin	Custom
Purpose	**SR-FR-11**, **SR-FR-12**, **SR-FR-14** and **SR-NF-20**
Function	0.Enables volume control. 1.Controls material to use on animation. 2.Defines by-default MIDI mappings. 3.Handles listening MIDI Channel 4.Enables remapping by creating submenus associated to each specific Drum piece.
Type	Executable
Dependencies	**Audio Mixer** (C6) **Interface ConfigVolume**: allows access to the AudioMixer utilities, allowing applying FX and controlling overall volume, **Audio Source** (C4) **Interface AudioPreview**: allows sound preview functionality as well as reassignment. **Drumset Controller** (C3) **Interface DrumsetInfo**: allows configuration of drum pieces, customization of noteNumber to sound mappings. **Game Controller** (C2) **Interface DrummMappingSet**:allows access to central mapping dictionary, accessible throughout the application.
Offered interfaces	**Interface Configure**: allows activation of configuration component , allowing for customization of sounds, CC knob used, etc.
Target build	Modest Build

Table 4.6: Component 7 Specification Modest Build

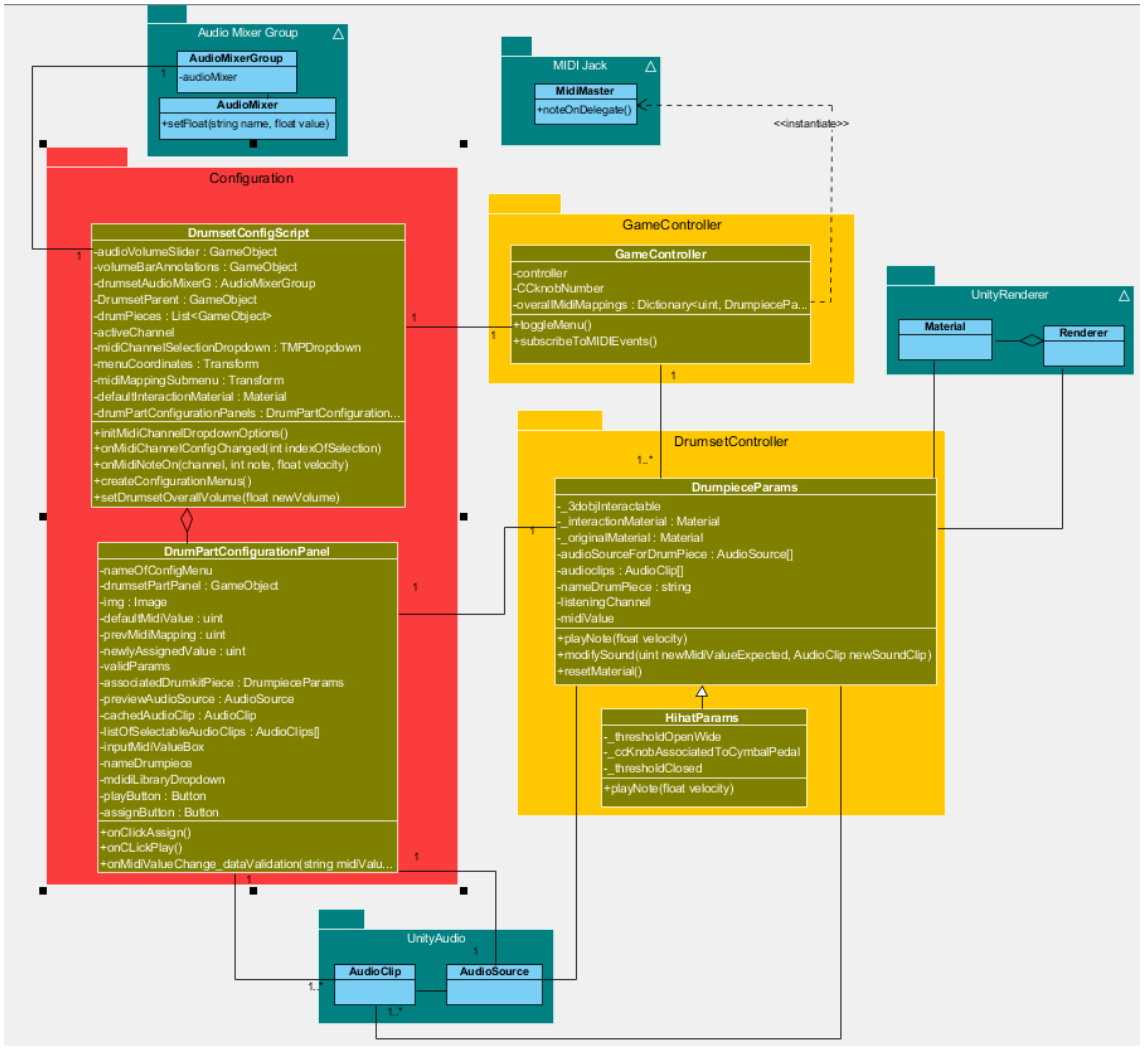


Figure 4.11: Class Diagram Modest Build

4.2.2.3.2 User Interface and Visualization

Several UI aspects were revisited in this second build, developing different improved solutions that enabled more customisation from the point of view of the user. Based upon Ge Wang's design principles, we introduced the following features in this build: in order to "Invite the eye" (3^{rd} principle) and provide an "aesthetic" (10^{th}) experience, we created a brand new model of the drumset, with a lot of work put on it to ressemble a real one as accurately as possible.
A configuration menu with lots of visual elements was also provided (see Figure 4.12), allowing users to modify which sounds are triggered when certain MIDI input is received. They can also modify the output sound amplitude for the whole drumset using a slider and they can choose a Channel (from 1-16) in order for the system to listen to MIDI input solely from that channel.

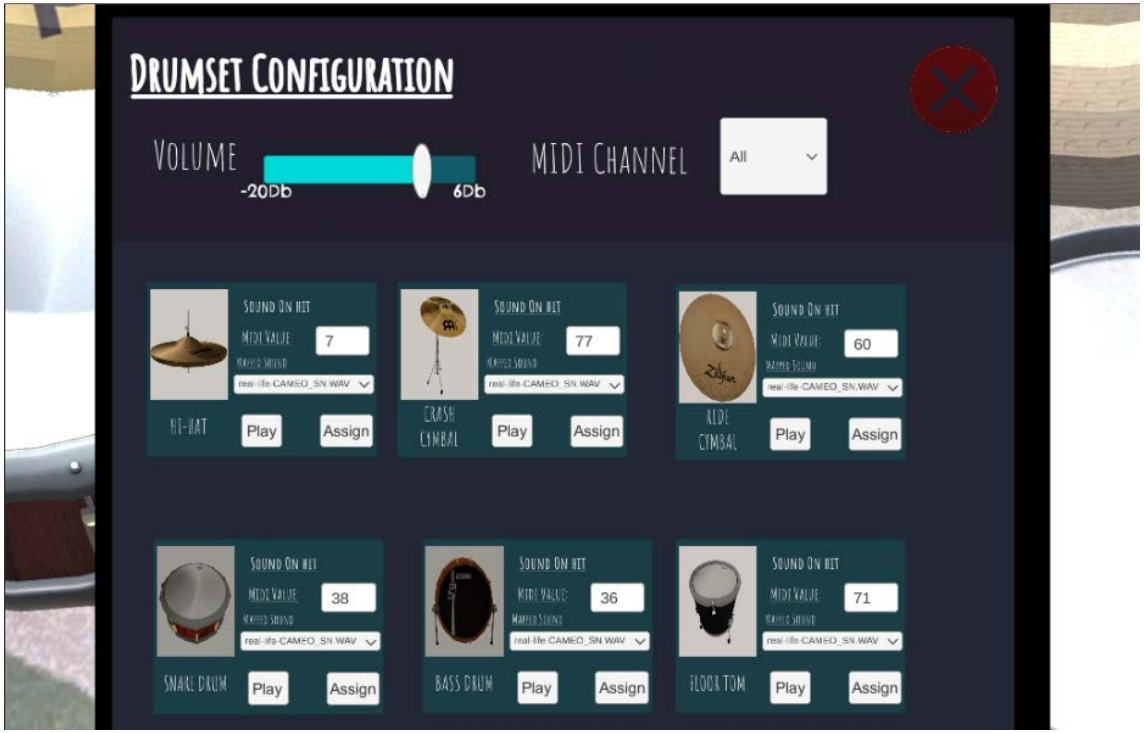


Figure 4.12: Configuration menu implemented in Modest Build using Unity

Note that, as already implemented in the Bare Bones build, the system complies with the 8th principle,"Animate"), since on interaction materials from Drum-set pieces turn red and go back to normal when the sound is paused or over. In addition, this implies the application of using "Graphics to reinforce physical interaction" (the 6th principle) and "Simplify" (the 7th), since the design was made so that the user can very easily map their actions to effects within the virtual environment, and no really complex animations are executed, allowing the user to focus in playing and generating sounds. This is the chief purpose of the system.

Figure 4.13: Drummer's viewport

Chapter 5

Evaluation

In this chapter, we evaluate conformance with respect to the system requirements defined in Section 3.3, in order to show evidence of the correctness of the solution. Firstly, various tests will be described along with their output in section 5.1, and later, a traceability matrix plotting System requirements against Tests performed will be provided to assure completeness (see 5.2.

longtable

5.1 Requirements fulfilment analysis

In order to describe tests formally, a template will be used to structure tests, so that no information strictly required is accidentally left out of the test definition. This approach to testing allows for an easy traceability of the System-Requirements whose compliance is to be assured by performing and verifying these tests. The template covers the following aspects of each test:

- **ID**: alphanumeric string that univocally identifies the test. It follows the naming pattern TX, where X is a number that increases as new tests are performed.
- **Name**: a sentence summarising the test at the conceptual level.
- **Requirements covered**: Lists the number of requirements that whose compliance is to be addressed by means of this test.
- **Required set-up**: Often, tests require a set of assemblies or some preparation before being carried out. This field summarises the fundamental steps to perform prior to testing.
- **Test description**: an in-depth description of the actions to be carried out for testing is provided along with some general considerations about what the test aims to achieve.

- **Expected Results**: for each described action within the "Test Description" field, a set of results are to be observed, in order to assess whether the test can be concluded to be either successful or failure.
- **Observations to make**: list a set of side effects of the test that are as well required for verification.
- **Result**: has to possible values, Verified or Failed. This field is to be filled once the test has been carried out, stating the final outcome of the testing phase for each specific test.

Table 5.1: Test Cases template

Test Cases Template	
Name:	e.g. Descriptive brief name for the test
Requirements Covered:	e.g. SR-NF-01, SR-FR-02, SR-FR-04, SR-NF-03, SR-FR-05, SR-NF-04, SR-FR-21, SR-FR-22, SR-NF-05, SR-FR-06, SR-NF-10, SR-NF-12, SR-NF-13
Required Set-up	e.g. The user shall execute the program and configure it to listen to all MIDI ports. They shall connect the MIDI controller to the host computer and place a practice pad next to them in order to test hit-based interaction.
Test description: Expected Results:	e.g. Here you may find the objective of the test briefly detailed. Then, a set of actions to be performed will be enumerated as follows: A1) A user will perform a soft hit with the left-hand drumstick on a practice pad.
Observations to make:	Any extra observations to make to assess test verification.
Result:	Verified ✓ | Failed ✗

T1	
Name:	Limb-isolated operation of gesture controller Unit test
Requirements Covered:	SR-NF-01, SR-FR-02, SR-FR-04, SR-NF-03, SR-FR-05, SR-NF-04, SR-FR-21, SR-FR-22, SR-NF-05, SR-FR-06, SR-NF-10, SR-NF-12, SR-NF-13, SR-NF-14, SR-NF-15, SR-NF-16, SR-NF-17, SR-NF-18, SR-FR-15, SR-NF-22, SR-FR-17, SR-FR-03.
Required Set-up	The Arduino-based gestural controller will be connected via USB to a host computer running Windows 10 and *Cubase LE AI Elements 8 64bit*. *Hairless MIDI* and *loopMIDI* will run on the host computer as well, configured as follows in order to perform the tests: *LoopMIDI* configuration: Create a new loopback MIDI port, giving it a name we will be using later into Hairless MIDI. e.g. "DRUMAR". Hairless MIDI configuration: • A baud rate of 256000, • A length of data bits of 8 • No parity bits used • Use one stop bit in order to separate mesages in the received stream • No control flow. Assign the loopback MIDI port created previously to the MIDI Out dropdown Menu, in order to send data through that port from the Arduino Serial Port. Enable Debug Logging in Hairless MIDI to display a message when MIDI input is received, with timestamps.

Continuation of T1 (table 5.1)	
T1	
Name:	Limb-isolated operation of gesture controller Unit test
Test description:	The test will consist on the following actions: A1) A user will perform a soft hit with the left-hand drumstick on a practice pad. A2) A user will perform a hard hit with the left-hand drumstick on a practice pad. A3) A user will use an actual bass drum pedal to hit a piezo sensor placed under a piece of foam attached to a bass drum practice pad. A4) A user will perform a hit with the right-hand drumstick on a practice pad without using their foot to cover the LDR embedded within the physical left-foot pedal. A5) A user will perform a hit with the right-hand drumstick on a practice pad while completely occluding the LDR embedded within the physical left-foot pedal.

Continuation of T1 (table 5.1)	
T1	
Name:	Limb-isolated operation of gesture controller Unit test
Expected Results:	A1) - Hairless MIDI will print out a message notifying the user about a newly received MIDI message with note Number X and a small velocity value in the range of [0-127]. - A snare drum sound with a medium perceptible volume will be generated as output from the host computer's speakers. A2) - Hairless MIDI will print out a message notifying the user about a newly received MIDI message with note Number X and a small velocity value in the range of [0-127]. - A louder snare drum sound will be generated as output from the host computer's speakers. A3) - Hairless MIDI will print out a message notifying the user about a newly received MIDI message with note Number X and a small velocity value in the range of [0-127]. - A kick-drum sound whose volume is proportional to the velocity received as input from the MIDI message will be generated. A4) - The user shall be able to observe the green led attached to the physical left-foot pedal is off, since occlusion will not be taking place. - Hairless MIDI will print out a message notifying the user about a newly received MIDI message with note Number X and a velocity value that varies depending on the strength of the hit within the interval [0-127]. - An open hi-hat sound will be generated as output from the host computer's speakers. A5) - The user shall be able to observe the green led attached to the physical left-foot pedal is off, since occlusion would be happening. - A set of CC messages will be displayed in the Hairless MIDI windows showing a decreasing Continuous Control value that approaches 0 as complete occlusion is completed. - When the hit occurs, Hairless MIDI will print out a message notifying the user about a newly received MIDI message with note Number X and a velocity value that varies depending on the strength of the hit within the interval [0-127]. - A closed hi-hat sound will be generated as output from the host computer's speakers.

Continuation of T1 (table 5.1)	
T1	
Name:	Limb-isolated operation of gesture controller Unit test
Observations to make:	- The red LED, attached to the piezo sensor associated to the bass drum shall be on all the time, from the moment when the Arduino is connected and on. - No duplicated messages shall be produced when interaction occurs, that is to say, only actual hits shall be recognised as interaction and have an effect in the sound-generation subsystem. - The third-party software shall perform intermediary operations to allow receipt of MIDI messages and sound generation using Cubase with the Addictive Drums 2 VST.
Result:	Verified ✓

T2	
Name:	Basic Sound-generation subsystem Unit Testing
Requirements Covered:	SR-FR-03, SR-FR-19, SR-NF-21, SR-NF-20, SR-NF-19, SR-FR-16, SR-NF-08, SR-FR-08, SR-FR-10, SR-NF-06, SR-FR-09, SR-FR-07, SR-NF-02, SR-FR-01, SR-NF-25.
Required Set-up	The Unity-based program will be tested using a Novation MIDI keyboard to test sound generation and visual animation features of the system. This approach is chosen to assure compatibility with a general-purpose MIDI controller. Pre-configuration and execution of Hairless MIDI and loopMIDI are necessary and require the same steps as in T1 (see table 5.1).
Test description:	The test will consist on the following actions: A1) A user will press key number 35 (Acoustic Bass Drum in GM) A2) A user will press key number 38 (Acoustic Snare in GM) A3) A user will press key number 42 (Closed Hi-hat in GM) A4) A user will press key number 41 (Low FLoor Tom in GM) A5) A user will press key number 46 (Open Hi-hat in GM) A6) A user will press key number 51 (Ride Cymbal in GM) A7) A user will press key number 49 (Crash Cymbal in GM) A8) The user will move the mouse around once the sound-generating program has launched.
Expected Results:	For actions A[1-7] : - The 3D representation of the corresponding sound should momentarily turn red while the sample sound is playing. - A sound associated to the hit drum piece should be triggered with a volume which will be proportional to the exerted force on the MIDI keyboard key pressed. A8) The camera view will change depending on the direction of the mouse movement, allowing the user to observe the virtual environment as if they were changing their gaze direction while sat on the drum throne.

Continuation of T2 (table 5.1)	
T2	
Name:	Basic Sound-generation subsystem Unit Testing
Observations to make:	- No sounds shall be generated unless the user presses a key, as in the case of Actions [1-7]. - The aspect of the 3D represenations within the sound-generation subsystem shall go back to normal once sounds stop playing.
Result:	Verified ✓

T3	
Name:	Physical metrics of MIDI Controller subsystem
Requirements Covered:	SR-NF-24, SR-NF-23
Required Set-up	N/A
Test description:	The MIDI controller subsystem is required to comply with a set of portability constraints, which are to be assessed by means of this tests. The parameters to be assessed are weight, width and depth, which are expected to be 4kg, 84cm and 72 cm respectively; in order to check compliance with these requirements a user will be ask to perform the following actions: A1) The user will lay the whole subsystem on the ground, involving permanent and portable connectors (such as USB-to-USB cables), along with the components of the tangible interface. The use will take measurements using a measuring tape to determine whether the dimensions are lower than the expected ones. A2) The user will weight the whole subsystem using a standard scale to determine compliance with the weight requirements.
Expected Results:	A1) The subsystem's dimensions do not surpass those specified in the requirements. A2) The weight of the subsystem is lower than 4 kg.
Observations to make:	N/A
Result:	Verified ✓

T4	
Name:	Concurrent multi-port interaction with Sound-generation subsystem
Requirements Covered:	SR-FR-01, SR-NF-01,SR-NF-04, SR-NF-02, SR-NF-07, SR-NF-08, SR-FR-08, SR-FR-09, SR-NF-06, SR-FR-10
Required Set-up	In order to determine whether MIDI from two different virtual or hardware MIDI controllers can be correctly processed by the sound-generation subsystem, concurrent (potentially) parallel input is provided using both the custom MIDI controller and the aforementioned Novation MIDI Keyboard (see T2; table 5.1). - Hook both instruments up to the host computer via USB. - Open Hairless MIDI and loopMIDI and set them up as specified in T1, in order to redirect Serial input from the Arduino properly to the sound-generation application. - Make sure Automap 4 is running and creating its own MIDI port. - Run the sound-generation application in Unity. - Open the MIDIJack terminal window so that it shows information regarding input messages.
Test description:	The actions to be performed in this test are: A1) A user will press key number 38 (Acoustic Snare in GM) on the Novation MIDI keyboard while hitting the piezo sensor that is mapped to the bass drum note number (35) in the Arduino-based instrument.

Continuation of T4 (table 5.1)	
T4	
Name:	Concurrent multi-port interaction with Sound-generation subsystem
Expected Results:	A1) - Hairless MIDI will print out a message notifiying the user about a newly received MIDI message with note Number 35 and a small velocity value in the range of [0-127]. - Check the logs in the MIDI Jack Window to see the received messages in the message History. There should be 2 messages only and at least two MIDI ports shall display at the top, the address of the message should correspond to that identifying a different MIDI port. - An Acoustic Snare drum sound shall be triggered and the snare drum aspect shall change during the time the sounds is playing. - A Bass drum sound shall be triggered and the Bass drum aspect shall change during the time the sounds is playing.
Observations to make:	Sound generation should only happen on purposeful interaction and no messages shall be exchanged between controller and sound-generation subsystem while the actions are not being performed.
Result:	Verified ✓

T5	
Name:	CC output test Gesture Controller Unit Test
Requirements Covered:	SR-NF-01, SR-NF-02, SR-FR-04, SR-FR-04, SR-NF-03, SR-FR-05, SR-NF-04, SR-FR-22, SR-FR-21, SR-FR-06, SR-FR-07, SR-NF-12, SR-NF-13,SR-NF-15, SR-NF-16, SR-NF-17, SR-NF-18, SR-FR-15, SR-NF-22, SR-FR-17, SR-FR-03
Required Set-up	CC messages are known to easily overflow the Serial bus if not correctly implemented, in this test we use *Addictive drums 2* to test the CC output as suitable for control of a virtual instrument in an isolated fashion. This tool was chosen because of the great User Interface it offers, allowing to visualise the state of the hi-hat in a continuous bar that maps to hi-hat aperture within the VST. In order to set up the testing environment, we need to open *Hairless MIDI* and *LoopMIDI* and configure them as in T1 (see table 5.1). Then, we must open the Standalone version of *Addictive Drums 2* and open the Audio & MIDI Setup menu. Then, we must select the DRUMAR port as active MIDI input. Then, go to the "?" option and "Map" Window, a CC Hihat map will be shown, displaying CC values received. Depending on the CC value received, when hitting with our right-hand drumstick, we shall produce one sound or another (all of them being hi-hat sounds).
Test description:	The actions to be performed in this test are: A1) Keep the Pedal completely without occlusion and perform a hit in the practice pad. A2) Place your left foot over the pedal, completely occluding the LDR and perform a hit in the practice pad. A3) Raise your toes up without leaving the pedal with your heel and perform a hit in the practice pad. A4) Alternate between foot position described in A2 and A3 and observe the received input in the host computer.

Continuation of T5 (table 5.1)	
TX	
Name:	CC output test Gesture Controller Unit Test
Expected Results:	A1) - No CC messages shall be displayed in Hairless MIDI unless the light reaching the LDR sensor decreases due to environmental light change. - An open hi-hat sound shall be generated by Addictive Drums 2 while the CC marker shows the indicator at a high position within the CC value plotter. A2) - A set of CC messages will be displayed by Hairless MIDI, decreasing in CC value as occlusion is performed on the LDR. - A closed hi-hat sound shall be generated by Addictive Drums 2 while the CC marker shows the indicator at a low position within the CC value plotter. A3) - A set of CC messages will be displayed by Hairless MIDI, increasing in CC value as less occlusion is performed on the LDR. - A partially opened hi-hat sound shall be generated by Addictive Drums 2 while the CC marker shows the indicator at an intermediate position within the CC value plotter. A4) - A set of CC messages will be displayed by Hairless MIDI, increasing and decreasing in CC value occlusion varies (the more occlusion the lower the cc value). - The CC marker in Addictive drums shall show the indicator varying on an intermediate position within the CC value plotter.
Observations to make:	- No sounds shall be generated if no interaction from any of the user's limbs happens. - When occlusion is performed on the LDR, the green LED attached to the left-foot pedal will shine. - The volume of the sounds generated will depend on the strength of the performed hits.
Result:	Verified ✓

T6	
Name:	Parallel hitting gesture controller test
Requirements Covered:	SR-NF-09, SR-NF-01, SR-FR-05, SR-NF-04, SR-FR-21, SR-FR-06, SR-NF-10, SR-NF-12, SR-NF-13, SR-NF-14, SR-NF-15, SR-NF-16, SR-NF-17, SR-FR-15
Required Set-up	The set up is the same as in the case of (see 5.1), running Addictive Drums 2 and *Hairless MIDI+LoopMIDI*.
Test description:	The purpose of this test is guarantee input from all limbs can be effectively gathered without losses from the performer's perspective. The idea is to get a user to perform hits using both hands and the bass drum pedal at the same time 50 times, in order to assess whether the gestural controller system is able to generate 3 midi messages at least 80 percent of the attempts. The actions to be performed in this test are: A1) A user shall perform all three hits simultaneously, triggering three sounds at the Addictive drums 2 end.
Expected Results:	A1) <u>Do 50 times:</u> - Three messages shall be displayed by Hairless MIDI, including respectively number 35 (Acoustic Bass Drum in GM), 38 (Acoustic snare) and 7 (hi-hat shaft in Addictive Drums 2). - Three sounds associated with the drum set pieces that map to those note Numbers shall be generated.
Observations to make:	- Count the number of times that all threee messages are properly sent to the Addictive drums end and compute the percentage out of the 50 attempts. If it surpasses the 80% then the system is compliant with this set of requirements.
Result:	Verified ✓

T7	
Name:	Performance test
Requirements Covered:	SR-FR-05, SR-NF-04, SR-FR-21, SR-FR-06, SR-NF-10, SR-NF-11, SR-NF-12, SR-NF-13, SR-NF-14, SR-NF-17, SR-FR-15
Required Set-up	In order to test that we use Hairless MIDI debug terminal, that timestamps MIDI messages with the number of milliseconds from the start of the program, allowing to easily compute the interval between two consecutive MIDI messages received.
Test description:	This test measures responsiveness of the gesture controller subsystem, which is to allow 2 consecutive hits from any limb within the 68ms timespan. The actions to be performed in this test are: A1) Perform two consecutive hits leveraging drumstick rebound, using the left-drumstick. A2) Perform two consecutive hits leveraging drumstick rebound, using the right-drumstick. A3) Perform two consecutive hits to the piezo sensor associated to the bass drum (leveraging pedal rebound).
Expected Results:	For all of the actions, it shall be possible to execute a two hits with a separation of around 68ms or less with the system producing output messages and sounds; For each action, the user must compute the interval between the receipt time of the two last consecutive MIDI NoteOn messages to compare it with 68ms.
Observations to make:	N/A
Result:	Verified ✓

T8	
Name:	Customisation functions test of the Sound-Generation subsystem
Requirements Covered:	SR-NF-02, SR-NF-02, SR-FR-02, SR-FR-07, SR-FR-08, SR-FR-09, SR-NF-06, SR-NF-06, SR-FR-10, SR-NF-08, SR-FR-11, SR-FR-12, SR-FR-13, SR-FR-14, SR-NF-17, SR-FR-16, SR-NF-19, SR-NF-20, SR-NF-21, SR-NF-27, SR-FR-20
Required Set-up	The Novation MIDI controller will be will be used to carry out this test, in order to simplify the process, requiring simply to press different keys once mappings have been altered. This saves time regarding the need to alter mappings in the custom MIDI controller subsystem if we wanted to use an integrated approach to testing.
Test description:	This test assesses the correctness of the configuration features provided by the sound-generation subsystem created in this dissertation; that is, mapping alteration, sound mapped-change, active channel, sound preview, sound assignment and volume change. The actions to be performed to test out all this functionalities are : A1) - Press "Esc" to open the Drumset Configuration Menu. - Hit key number 35 in the piano keyboard (mapped to acoustic bass drum by default). - Move the slider to the left and play the note again. The volume shall be lower now. - Move the slider back to its original position and play the note again. The volume should increase. - Move the slider to the right and play the note again.The volume should increase even more. - Play the note several times while moving the slider value, observe that changes in volume are indeed performed. A2) - Configure the Novation MIDI keyboard to send messages targeted at Channel 1 only. - press key number 38 (mapped to the acoustic snare). Observe that a sound is generated. - Change the active channel from "All" (the default) to 9. Play the same note again. No sound shall be generated. - Change the active channel from 9 to 1. Play the same note again. A sound shall be generated. A3) - Press key number 41 (Low FLoor Tom in GM) in the Novation MIDI keyboard. - Choose a new sound to be mapped and press "Play". The new sound should be generated for preview. - Choose a different channel to be mapped and press "Play". A different sound should be generated for preview. - Change the MIDI value to number 1 and Click assign. - Press now note number 1 on your MIDI keyboard. You should hear the newly assigned sound and see the associated drum piece turn red accordingly.

Continuation of T8 (table **??**)	
T8	
Name:	Customisation functions test of the Sound-Generation subsystem
Expected Results:	A1) - The menu will display, showing the default configuration values for each drum piece, as well as volume and active channel. - Volume will change as expected and the change will be immediately executed. A2) - The system shall be able to generate sounds only when MIDI messages are set to be received by the Channel specified as "Active channel" or in the special case of active channel = "All". A3) - MIDI preview shall allow preview of a list of predefined sound samples. - Assignment of sounds and remapping shall work as expected, having a new note number assigned to the interaction with a given piece of the simulated drums.
Observations to make:	N/A
Result:	Verified ✓

T9	
Name:	Integration Test
Requirements Covered:	SR-FR-18, SR-FR-01, SR-NF-01, SR-NF-02, SR-FR-02, SR-FR-04, SR-NF-03, SR-FR-05, SR-NF-04, SR-FR-21, SR-FR-22, SR-NF-05, SR-FR-06, SR-FR-07, SR-FR-08, SR-FR-09, SR-NF-06, SR-FR-10, SR-NF-12, SR-NF-13, SR-NF-14, SR-NF-17, SR-NF-18, SR-FR-15, SR-FR-16, SR-NF-19, SR-NF-20, SR-NF-21, SR-NF-22, SR-FR-17, SR-FR-18, SR-NF-25, SR-FR-03
Required Set-up	The set-up for the test requires T1 configuration of *LoopMIDI* and *Hairless MIDI* (see table 5.1), and execution of the sound-generation subsystem once those have been configured and no other sequencer program is running.
Test description:	This test aims to demonstrate the correctness of the solution, carrying out a sect of actions that involve both custom subsystem (the gesture controller and the sound generation subsystem). The actions to be performed in this test are: A1) The user will perform a hit to the piezo sensor attached to the bass drum practice pad (of the gesture controller). A2) The user will perform a left-drumstick hit to the practice pad (of the gesture controller). A3) Without occluding the LDR of the left-foot pedal, the user will perform a hit with the right hand-drumstick. A4) Completely occluding the LDR of the left-foot pedal, the user will perform a hit with the right hand-drumstick. A5) Partially occluding the LDR of the left-foot pedal (heel down), the user will perform a hit with the right hand-drumstick.

Continuation of T9 (table 5.1)	
T9	
Name:	Integration Test
Expected Results:	A1) - The aspect of the bass drum 3D representation will change momentarily and a sound with the appropriate timbre and volume proportional to the hit strength will be triggered. As the sound ceases, the aspect of the 3D object will go back to normal. A2) - The aspect of the snare drum 3D representation will change momentarily and a sound with the appropriate timbre and volume proportional to the hit strength will be triggered. As the sound ceases, the aspect of the 3D object will go back to normal. A3) - The aspect of the hi-hat 3D representation will change momentarily and a sound with the appropriate timbre (wide-open hi-hat) and volume proportional to the hit strength will be triggered. As the sound ceases, the aspect of the 3D object will back to normal. A4) - The aspect of the hi-hat 3D representation will change momentarily and a sound with the appropriate timbre (closed hi-hat) and volume proportional to the hit strength will be triggered. As the sound ceases, the aspect of the 3D object will back to normal. A5) - The green LED attached to the left-foot pedal shall light up. A6) - The aspect of the hi-hat 3D representation will change momentarily and a sound with the appropriate timbre (mid-open hi-hat) and volume proportional to the hit strength will be triggered. As the sound ceases, the aspect of the 3D object will back to normal.
Observations to make:	- No sound shall be generated unless interaction occurs.
Result:	Verified ✓

5.2 Evaluation Traceability matrices

This section plots Test cases against the Software requirements they check. Each "x" means this test checks for the fulfilment of the requirement in the same row.

	T1	T2	T3	T4	T5	T6	T7	T8	T9
SR-FR-01		x		x					x
SR-FR-02	x							x	x
SR-FR-03	x	x			x				x
SR-FR-04	x				x				x
SR-FR-05	x				x	x	x		x
SR-FR-06	x				x	x	x		x
SR-FR-07		x			x			x	x
SR-FR-08		x		x				x	x
SR-FR-09		x		x				x	x
SR-FR-10		x		x				x	x
SR-FR-11								x	
SR-FR-12								x	
SR-FR-13								x	
SR-FR-14								x	
SR-FR-15	x				x	x	x		x
SR-FR-16		x						x	x
SR-FR-17	x				x				x
SR-FR-18									x
SR-FR-19		x							
SR-FR-20								x	
SR-FR-21	x				x	x	x		x
SR-FR-22	x				x				x
SR-NF-01	x			x	x	x			x
SR-NF-02		x		x	x			x	x
SR-NF-03	x				x				x
SR-NF-04	x			x	x	x	x		x
SR-NF-05	x								x
SR-NF-06		x		x				x	x
SR-NF-07				x					
SR-NF-08		x		x				x	
SR-NF-09						x			
SR-NF-10	x					x	x		
SR-NF-11							x		
SR-NF-12	x				x	x	x		x
SR-NF-13	x				x	x	x		x
SR-NF-14	x					x	x		x
SR-NF-15	x				x	x			
SR-NF-16	x				x	x			
SR-NF-17	x				x	x		x	x
SR-NF-18	x				x				x
SR-NF-19		x						x	x
SR-NF-20		x						x	x
SR-NF-21		x						x	x
SR-NF-22	x				x				x
SR-NF-23			x						
SR-NF-24			x						
SR-NF-25		x							x
SR-NF-27								x	

Figure 5.1: SR against Test Cases Traceability matrix

Chapter 6

Project plan

6.0.1 Methodology selection concerns

This project has been carried out following the ESA standard targeted at small software projects [76]; A generic software life cycle as stated by ESA standard is composed by a set of 6 phases. Even though the phases of the software life-cycle are (to some degree) fixed, there are many different approaches to them, since projects vary in shape and necessities. For this reason, the present section presents some of the most usual approaches and justifies the selection of one to be applied to the planning and development of this dissertation.
It is worth summarising the software life-cycle phases for completeness in this document. In broad strokes, every software product is the result of the following subprocesses.

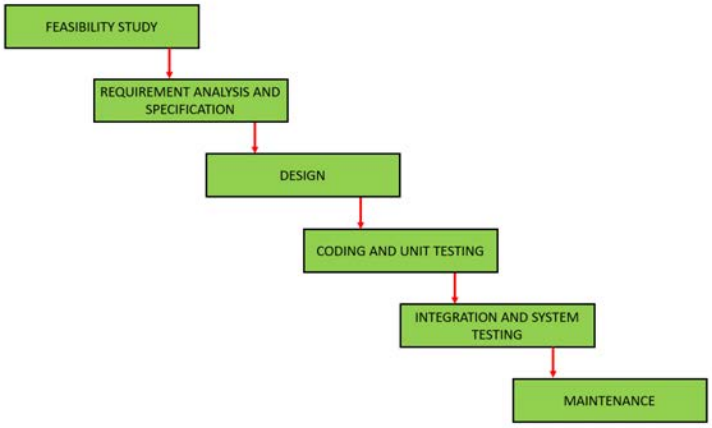
1. **UR Phase**: Definition of User requirements: also called the "problem definition phase" consists in the user of the system to be built discussing their needs along with the elicitator or developer (in case of very tiny projects) in charge of clearly writing a set of user requirements down, which serve as input to the whole development process. Those requirements are of two types, as described in Section 3.2; Capability requirements and Constraint requirements.
2. **SR phase**: Definition of System requirements: also called the "Problem Analysis Phase", consists in a thorough specification of what the system is to do, without using implementation terminology. They describe high-level essentials, making the system understandable as a unit. The documentation item associated with this phase can be found in Section **??** of this paper.
3. **AD phase**: Definition of architectural design. The developer is in charge of taking the System requirements and use them as an input to generate a physical model; that is, providing a solution for the proposed problem in implementation terms. This phase and the System Requirements phase are to be merged in order to save efforts according to the ESA recommendations targeted at small projects.
4. **DD phase**: Detailed design and production of the code. Thus, implementation along with code documentation. This phase is simplified and not formally

performed; that is, no document is generated for it according to the ESA recommendations for small projects.

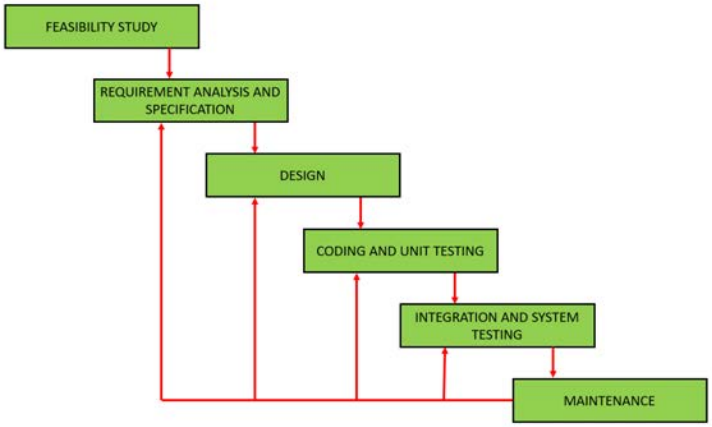
5. **TR phase**: transfer to the software to operations. In other words, software is set to function in the operational environment it was designed to work within.
6. **OM phase**: operations and maintenance. Any debugging that could not be performed in previous phases shall be performed so that the software keeps fulfilling the requirements that gave rise to the system in the first place. The ultimate evaluation will be carried out during this phase, where the product will be operating in the target environment.

These phases can be approached in various ways, among which we find the following:

- **The Waterfall approach**: as its name points out, the Waterfall proposal consists in a sequential execution of the aforementioned phases, allowing iteration over past phases for the sole purpose of error correction. It is the most straightforward approach for software development; say the most classical approach, but likely, the one used the least mainly due to the lack of flexibility and its poor applicability among real-world projects. It is more of a 'theoretically perfect' approach [77].
  As the original Waterfall approach is not practical, an improvement was performed by adding a feedback path from a phase towards its preceding phases, allowing for errors during the development. However, this model is still struggling with flexibility and a lack of feedback from the end user, since according to this approach, it is only once the system is completely built when the user can review the actual product beyond documents regarding requirements.
- **Incremental delivery approach**: This approach aims to modularize the process of development of the system, allowing for the three last phases to be carried out independently for each module identified in the Architectural design phase. In other words, this model, also known as "Successive version model", proposes the implementation of the core functionality and expanding upon the basic version of the system to be provided at the end of the process. Thus, by means of subsequent versions of the system, the features increase and are refined as new versions and feedback from the user is received. In this way, large systems can be easier to manage, and a set of incremental deliveries may be scheduled in case there are functions that require others to be implemented before they can be effective.
  This approach focuses in the short term planning targeted at completing the immediately next delivery, instead of trying to cover in the requirements the whole system precisely. There are several varieties of Incremental delivery; the Staged delivery model and the Parallel development model.
  - **Staged Delivery**: focuses on the construction of only one part of the project at a time; that is, it does not allow for parallel work on different components.
  - **Parallel Development Model**: in this variety several subsystems are developed concurrently, which in case there are enough resources, can lead to a reduced development time of the project overall.



(a) Original *Waterfall Model*

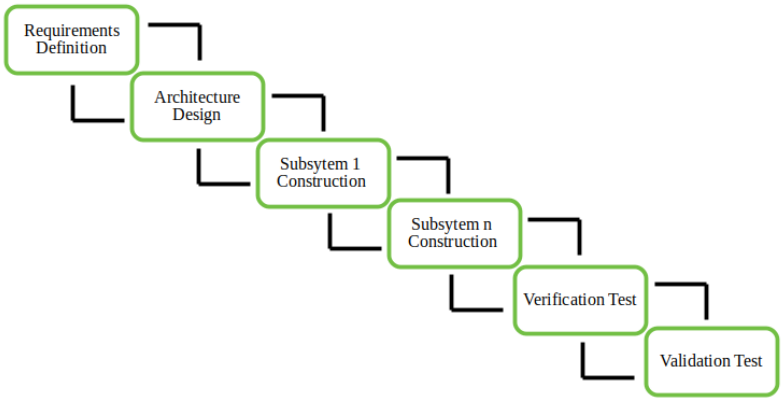


(b) *Iterative Waterfall Model*

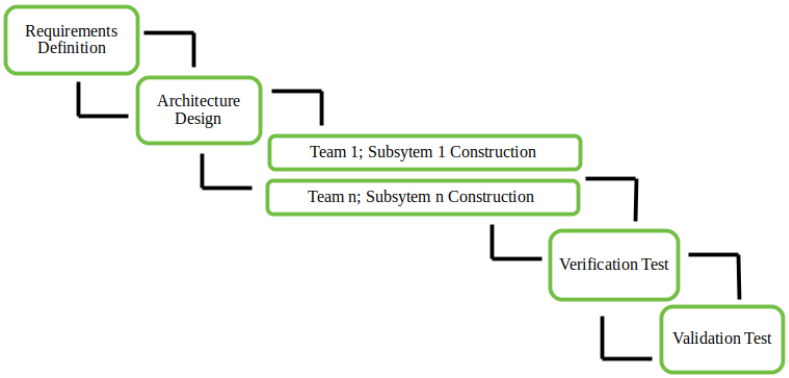
Figure 6.1: Software Lifecycle *Waterfall* approaches

The problem with Incremental delivery lies in the fact that incremental builds require extra testing to confirm the last release has not impaired functionality formerly tested. This, obviously has an impact in the cost of the software product as a whole, in terms of time (applying it to this project) and also regarding monetary cost; but its advantages are numerous, as we will describe shortly.

- **Evolutionary development approach** in this strategy, the idea to schedule a set of releases so that the development of the next can be performed using the experience from past deliveries. For every release, all phases of the life-cycle are carried out (all the lifecycle stages).
  The key aspect of the evolutionary development scheme is to recognise the priorities of the user and produce the fundamental pieces that are most valuable to the user and also can be developed with the minimal technical problems or expected delays. In such an approach, it is not strictly required to fulfil all requirements in each development cycle, but they shall be carefully stated so that the objectives whether fulfilled or not, can be identified and analysed appropriately.
  The representation of this model diagramatically consists in a spiral, indicating both stages and phases of progressive refinement of the product. Each new



(a) *Staged Delivery*



(b) *Parallel Incremental Delivery*

Figure 6.2: Incremental Delivery Lifecycle approaches

loop embodies an iteration over the product to be built, the radius of each loop reflects the expenses of the project whereas the angular dimension is a measure of progress within the project. Four quadrants can be spotted in Figure 6.3 each one relates to a stage of the process which internally involve the phases described at the beginning of this section. These quadrants are:

1. Objectives determination and leveraging of alternative solutions: involves the elicitation phase (UR and SR), depicting the problem and analysing possible solutions (AD phase).
2. Identify and resolve risks: Different alternatives are compared and the best proposal is prototyped (DD phase) in order to estimate risks and evaluate feasibility.
3. Development of next version of the product: Involves actual implementation of the code, testing and verification.

4. Review and plan of the next phase: evaluation happens because the software is then accessible to the customer (TR phase) and then a new loop starts (OM phase, maintenance involves the same phases over again).

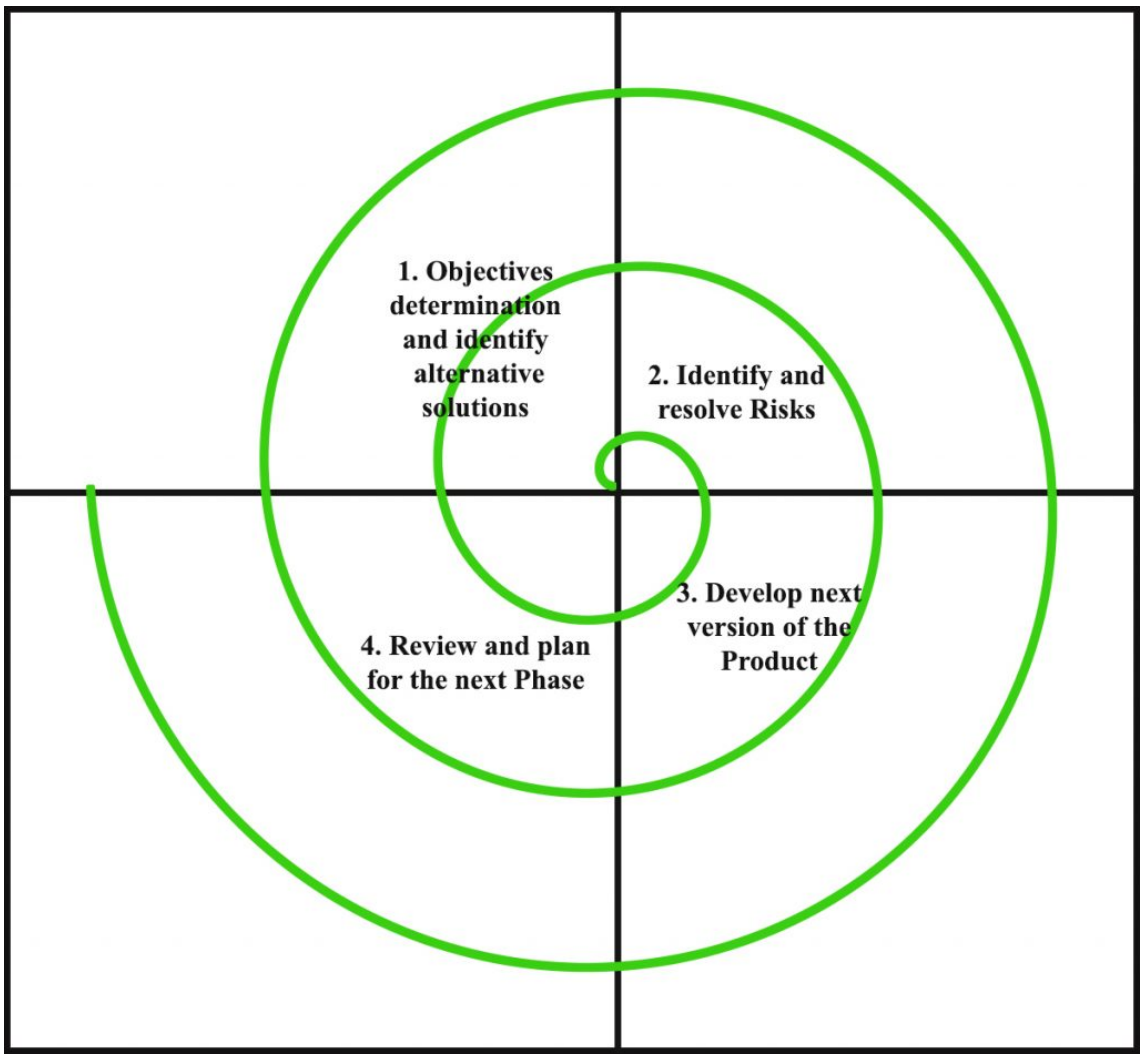


Figure 6.3: Spiral model

In a sense, this model leverages the Incremental Delivery approach since each new iteration generates a new version of the system with extended functionality. However, this model exceeds the advantages of the previous one in terms of Risk analysis, which is a crucial aspect to be taken into account when aiming at developing a product in an area relatively new in the IT world; it is important to know whether the conceived product can eventually be built or not before starting the development process when using Incremental delivery, whereas the Spiral model is much more flexible in this aspect [78].

Once these three methodologies have been described, the advantages and drawbacks of them will be discussed to finally choose the most suitable methodology for the present project.
On the one hand, the *Waterfall approach*, even though conceptually simple, doesn't seem to fit inexperienced developers and a project of the features and setting of the

present one, since the Waterfall model requires to provide a thorough description of the system, along with its design without taking into account the fact that certain requirements may not even be achievable in practice; that is, unfeasibility is not even considered as an option, since such evaluation takes place at the very beginning of the project and thus, it is assumed an infinite expertise of the developers, which are supposed to be able to predict the difficulties that are to be found during the software lifecycle.
In addition, the *"original" Waterfall model* has no feedback path; that is, no error shall be committed by the developers during any phases (which is completely impossible), since no error correction paths (no iterations) are planned.
In the case of the *Iterative Waterfall* improvement, it keeps requiring too much expertise from developers from the very beginning.
Flexibility is another issue, since this model requires requirements to be clean, complete and constant along the whole software lifecycle, so no changes to the requirements can conceivably be made to fix some major issue that the customer really needs to be addressed by means of the software.
Finally, efficiency constitutes another drawback to be conveyed in this dissertation, since development time is very important and the prohibition of overlapping phases proposed by the Waterfall model may be to costly in terms of both time and monetary costs.

In contrast, the *Incremental delivery approach* is a good fitting strategy regarding its stress on modularization of the implementation and subsequent phases, specially considering the shape of its Parallel Development variety; which at first glance seems to adapt really well to the easily spotted components of basically any MIDI based instrument (a hardware MIDI controller -a control system, hardware based - and a video-game like interface -implemented as Software-), both of which can be implemented concurrently following this approach and partial deliveries are proposed, which allow for completion of independently interfaced components.
However, it is likely to hinder the project, requiring much more time, to create a functional whole that meets the expectations of the user, which, indeed, may be too high.
What's more, **this model is said to adapt very well to projects using new technology** and uses the divide an conquer approach for splitting the major problem into smaller tasks that are easier to handle from the development point of view.
It is worth mentioning the necessity of stating requirements for each new version as clearly as possible up-front; that is, before any development starts; consequently, in the case we do not have a clear idea about what our product would be like, a different model would be a better choice.

The leverage of the *Evolutionary development* approach is its suitability for a variety of technical difficulties certain requirements the present project involves. The reason for this is its stress on Prototyping and flexibility.
In contrast, since the feasibility to fulfil some requirements in a timely manner may actually depend on a third party, the availability of future technology, or the additional learning time due to the inexperience of the developers in the area, the author of this paper believes this strategy does not fit well the project.
The reason for this is that lies upon risk analysis, which is a topic that requires a level of expertise that a Computer Science undergraduate cannot offer, and there-

fore the effort required to accurately assess the feasibility of the project is likely to become excessive and consequently, this methodology was eventually discarded. Neither is it recommended for small projects as this one, not to mention its complexity surpasses all the other presented models, owing to the fact that the phases are not fixed from the beginning and planning is hindered.

In conclusion, due to the fact that we have isolated wholes that can be spotted straight away (Hardware Interface and VR part) the development will take advantage of the *Incremental delivery* approach and leverage the prototyping stress made by Evolutionary development, in order to evaluate feasibility of the imagined solutions before committing to finish the whole project, which is aimed at providing with a functional whole as opposed to a set of erratic development attempts that are very likely to occur due to the novice status of the author of this dissertation. Last but not least, as we mentioned earlier, this project will be prototyping-intensive, so that precisely will constitute a chief practice and will be used first to define better the system to be built (as a target user and programmer simultaneously). This approach is referred often as ***Exploratory prototyping*** in the IT framework.

6.0.2 Methodology depiction

As stated in the previous section, we decided to combine the Incremental delivery and Exploratory prototyping approaches to the software life-cycle. Thus, the project leverages both the try-and-fix features of Incremental delivery and the constant feedback and refining from the user of the system enabled by Evolutionary development. Next, we will elaborate on how this methodology was carried out along this project. The following phases were performed in order for the project to be built according to the basics of Incremental delivery:

1. **Requirements gathering phase**: involves both the User Requirements and System Requirements phases of the ESA-proposed life-cycle. All the requirements for the immediate next delivery were created and marked as such, optional or desirable requirements were be left for implementation on subsequent iterations on the project or implemented in the case there was time to perform more than one iteration timely before the delivery of this project. Major independent components were identified and possible solutions to be prototyped were elaborated.
2. **Divisions of tasks for implementation**: **Two builds** were scheduled per major component, one involving a functioning whole with very basic functionality, referred to as Bare-bones build; and another one known as Modest build, including more advanced functionality, ergonomics and usability in general. Both builds were planned and are shown in the Gantt diagram shown in Figure 6.4. Further decomposition of tasks can be found there and within section 6.1, regarding task subdivision. Plans for both builds were carried out at the beginning of the project, stating which milestones were to be achieved very roughly; then, those plans were refined once the requirements phase was over.
3. **Development**: After the planning has been performed and goals for each

build are clearly set, the code and hardware components of the system are to be assembled and created. This process is the result of the design and previous requirements definitions and involves testing to verify the requirements are finally fulfilled at the end of every iteration. Involves the DD and TR phases of the ESA-defined software life-cycle phases.

4. **Evaluation**: The user is presented the system and proposes solutions regarding future work and comments to be used in subsequent iterations. The VRMI (Virtual Reality Musical Instrument) will be evaluated from the performer's perspective, regarding task performance, playability, user experience and emotions resulting from the interaction with the software. Serafin et al.'s *"Virtual Reality Musical Instruments: State of the Art, Design Principles, and Future Directions regarding VRMIs"* and references in it can be used as guidance to the evaluation of the project [31].

6.1 Division of tasks and formal planning

In order to provide an estimation of the duration of the project at the very beginning as well as organising and simplifying the problem as a whole into more manageable tasks that can be addressed in a straightforward manner, a **GANTT diagram** was created (see Figure 6.4) to show on a glimpse the pending tasks, the relationships among tasks (including dependencies), the progress and milestones that shall be completed along the project life-cycle.
GANTT diagrams are a way of representing tasks or processes with respect to time firstly conceived by Karol Adamiecki and popularized in western Europe by Henry Laurence Gantt [79].
GANTT diagrams, along with other complementary diagrammatic tools (such as PERT) are a very useful tool for project managers and task management in general. These make even more sense when Resources are added to the tasks themselves. Resources represent workforce, that is, employees that can be assigned different tasks according to their role. As this is an individual project, it makes no sense to create a Resource or multiple resources associated to the author of this paper and assign him all the tasks of the project.

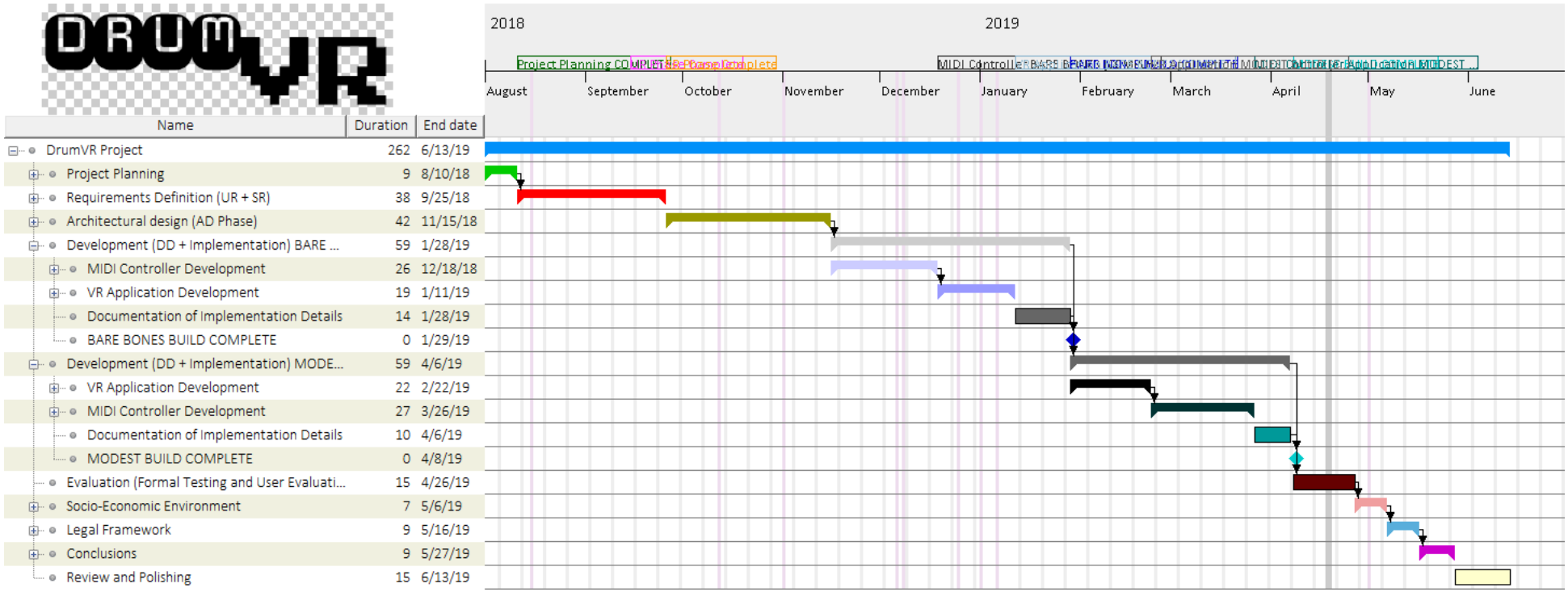


Figure 6.4: Project Plan GANTT Diagram

Finally, once the project came to an end, a table comparing estimates with respect to real begin and end dates can be found in Figure **??**, so that conclusions can be extracted to improve estimations in future similar projects. Note that diagrams were built using the free GPL-Licenced GanttProject[1].

The project was carried out in 10 months of work, formally starting in August 1^{st} 2018 and finishing by June 15^{th} not devoting full-time due to the fact that this application was built while working in a company and thus, has been built basically in scheduled spare time in order to guarantee its completion in June's deadline. For this reason, holidays are not taken as non-working days in GANTT diagrams, since those days were actually more work-intensive than working days, at least regarding the progress of this project.

[1]GanttProject: https://www.ganttproject.biz/

Phase	Planned			Actual		
Sub-Task	Begin Date	End Date	Duration	Start	End	Duration
Project Planning	8/1/2018	8/11/2018	9	8/1/2018	8/10/2018	8
Methodology Research	8/1/2018	8/3/2018	3	8/1/2018	8/3/2018	3
Methodology Reasoning	8/4/2018	8/6/2018	2	8/4/2018	8/6/2018	2
Subdivision of Tasks	8/7/2018	8/7/2018	1	8/7/2018	8/7/2018	1
GANTT Chart Creation	8/8/2018	8/10/2018	3	8/8/2018	8/9/2018	2
User Requirements	8/11/2018	9/14/2018	29	8/10/2018	9/5/2018	27
General Capabilities of the System	8/11/2018	8/13/2018	2	8/10/2018	8/10/2018	1
System Constraints	8/14/2018	8/16/2018	2	8/11/2018	8/11/2018	1
Description of Operational Environment	8/17/2018	8/18/2018	2	8/12/2018	8/12/2018	1
User Requirements Creation	8/20/2018	8/22/2018	3	8/13/2018	8/16/2018	4
Feasibility Analysis (State Of the Art)	8/23/2018	9/12/2018	18	8/17/2018	9/3/2018	18
Mark requirements for 1st Delivery	9/13/2018	9/13/2018	1	9/4/2018	9/4/2018	1
Traceability Matrix Creation	9/14/2018	9/14/2018	1	9/5/2018	9/5/2018	1
Use Cases	9/15/2018	9/17/2018	2	9/6/2018	9/7/2018	2
Function and Purpose	9/18/2018	9/19/2018	2	9/8/2018	9/9/2018	2
System Requirements	9/20/2018	9/25/2018	38	9/10/2018	10/18/2018	40
Functional Requirements	9/20/2018	9/24/2018	4	9/10/2018	9/14/2018	5
Non-Functional Requirements	9/20/2018	9/24/2018	4	9/15/2018	9/18/2018	4
Traceability Matrix UR/SR	9/25/2018	9/25/2018	1	9/19/2018	9/19/2018	1
Architectural Design	9/26/2018	11/15/2018	42	10/19/2018	12/4/2018	47
Pre-Design Research	9/26/2018	11/9/2018	37	10/19/2018	11/22/2018	35
Build Component Diagram	11/10/2018	11/12/2018	2	11/23/2018	11/27/2018	5
Build Sequence Diagram	11/13/2018	11/15/2018	3	11/28/2018	12/4/2018	7

Phase	Planned			Actual		
Sub-Task	Begin Date	End Date	Duration	Start	End	Duration
Bare Bones Development	11/16/2018	12/25/2019	45	12/4/2018	1/11/2019	39
MIDI Controller Development	11/16/2018	12/6/2018	26	12/4/2018	12/20/2018	17
VR Application Development	12/7/2018	12/25/2019	19	12/20/2018	1/11/2019	22
Documentation of Implementation Details	12/26/2019	1/28/2019	14	1/11/2019	1/22/2019	12
Modest Build Development	1/29/2019	4/6/2019	59	1/23/2019	3/28/2019	65
VR Application Development	1/29/2019	2/22/2019	22	1/23/2019	2/15/2019	24
MIDI Controller Development	2/23/2019	3/26/2019	27	2/16/2019	3/15/2019	28
Documentation of Implementation Details	3/27/2019	4/6/2019	10	3/16/2019	3/28/2019	13
Evaluation	4/8/2019	4/26/2019	15	4/4/2019	4/10/2019	7
Socio-Economic Environment	4/27/2019	5/6/2019	7	4/11/2019	4/18/2019	8
Legal Framework	5/7/2019	5/16/2019	9	4/19/2019	4/27/2019	9
Conclusions	5/17/2019	5/27/2019	9	4/27/2019	5/11/2019	15
Review and Polishing	5/28/2019	6/13/2019	15	5/23/2019	6/15/2019	24

Table 6.1: Planned Dates vs Actual Dates

Chapter 7

Socio-Economic Environment

This chapter reflects on the socio-economic concerns regarding the project, providing a description of the monetary costs as well as the impact the developed system may have on the society, end-users or other researchers of the Computer Science field. This chapter comprises two sections, expanding on these two main ideas.
In Section 7.1 the reader will be provided with the costs associated to the completion of this project, covering all the aspects involved, from human resources to hardware used to build the system itself, as well as the cost of licenses and other expenses that may be of interest of the reader. Later, in Section 7.2 a discussion will be presented, related to how this project may impact its surrounding social environment, describing the target market of the product detailed in this dissertation and debating about economic impact of it as well.

7.1 Project Budget

This section breaks down the set of costs derived from the project, covering different kinds of time and monetary resources that were required for its completion; described according to some categories.
Section 7.1.1 will focus on costs regarding software systems and licences required for production, as well as documentation of the project (as it is the example of Overleaf). Equipment-related costs will be presented in Section 7.1.2, while Human resources will be detailed later in Section 7.1.3. Finally, consumable costs along with indirect costs will be detailed in sections 7.1.4 and 7.1.5. At the very end of this passage, the total costs will be summarised, to give the reader a general idea regarding expenditure.

The costs along this section are computed as if the author of this document was working for a company during the whole time-span of the project, so expenses associated to goods are translated into costs in terms of amortisation. The formula to compute the amortisation costs associated is the following:

$$\text{Amortisation cost} = \frac{\text{Purchase price} \times \text{Percentage of amortisation}}{\text{months within a year}} \times \#\text{ of amortisation months} \tag{7.1}$$

For all the computations performed along this section, 10 months of work are assumed, starting from August 2018 until the end of May 2019. As recorded in Section **??**, 262 days of labour are considered, with an average of 3 hours and 30 minutes of work per day, already including one day off per week.

Total Project time	
Months	10 months
Days	262 days
Avg. hours per day	3.5 hours per day
Total labour time	890 hours

Table 7.1: Overall labour time required by the project

7.1.1 Software Resources

In this section we summarise the costs derived from software licenses and products that were utilised to complete this project. Knowing that the percentage of amortisation of Software resources according to Agencia Tributaria by 2018 [80] is 33%, we computed the costs associated to the software used along these 10 months.

Software Resources		
Software Product	Licence price	Amortisation
Windows 10 Education Edition	0.00€	0.00€
Unity Personal Edition	0.00€	0.00€
Visual Paradigm Community Edition 15.2	0.00€	0.00€
Microsoft Office Student Licence	0.00€	0.00€
GanttProject	0.00€	0.00€
Overleaf LaTeXeditor Online Student Licence	84.70€	23.29€
loopMidi	0.00€	0.00€
Hairless MIDI	0.00€	0.00€
Adobe Creative Cloud Student Licence	196.66€(per year)	54.08€
Cubase LE AI Elements (testing) 8	99.99€(permanent)	27.49€
Addictive Drums 2 (testing)	169.95€(permanent)	54.16€
Total:		159.02€

Table 7.2: Software Costs Summary

7.1.2 Equipment Costs

In this section we summarise the expenses regarding equipment, including computer systems, acquired hardware and so on. Note that amortisation percentage for IT equipment is 20%.
Table 7.3 shows the distribution of expenses regarding Computer systems used for programming, design and documentation (Equipment Resources 1) whereas the second part of the table shows the costs associated to the physical production of the system described in this document. The price for tools required for hardware as-

Equipment Resources (1)			
Computer systems used			
Equipment	Price	Cost per month (Amortisation)	Cost (10 months)
Personal Computer	499.99€	8.33€	83.33€
Total:	83.33€		

Equipment Resources (2)				
Product Hardware				
Item	Units	Price per unit	Total price	Cost (10 months)
Arduino Mega 2560 R3	1.00	33.00€	33.00€	5.50€
Pushbutton	12.00	0.15€	1.80€	0.30€
ELEGOO Breadboard	3.0	3.33€	9.99€	1.67€
USB 2.0 Type A Female Socket	8.00	1.19€	9.52€	1.59€
USB 2.0. Male to Male cable	4.00	1.44€	5.76€	0.96€
Grove Piezo Sensor	2.00	10.80€	21.60€	3.60€
Flexible Piezo Film (replacements)	2.00	3.52€	7.04€	1.17€
60m Cables	1.00	13.90€	13.90€	2.32€
Vic Firth 5A drumsticks	1.00	10.40€	10.40€	1.73€
Piezo sensor	6	0,74€	4.44€	0.74€
Light Dependent Resistor	4.00	0.16€	0.64€	0.11€
LED	3.00	0.59€	1.77€	0.30€
Resistor	6.00	0.02€	0.12€	0.02€
Potentiometer	1.00	0.61€	0.61€	0.10€
Total:			120.59€	20.10€

Table 7.3: Software Costs Summary

sembly and testing are depicted separately in table 7.4. Note that utensils of this kind have assigned a different amortisation percentage, which is 25%.

Tools				
Item	Units	Price per unit	Total Price	Cost(amortisation)
Multimeter	1.00	14.99€	14.99€	3.12€
Welder	1.00	19.99€	19.99€	4.16€
Total:			34.98€	7.29€

Table 7.4: Costs regarding Hardware assembly utensils

7.1.3 Human Resources

This section covers the cost of the project associated to labour itself. Computations have been performed using average salary values of technical engineers, that can be seen on 7.5. Different roles were fit by the author of this paper along its completion so wages vary shortly depending on the position.

Human Resources				
Role	Analyst	Designer	Programmer	Tester
Days of labour	164.5	147	287	126
Max salary/hour	21.33€	21.41€	21.27€	21.32
Min salary/hour	6.59€	6.81€	5.64€	5.85€
Base/hour	13.96€	14.11€	13.46€	13.59€
Total	9943.885€			

Table 7.5: Human Resources expenses

7.1.4 Consumable expenses

This section covers the office material and various consumables that were exhausted during the progress of the project, so in this case, no amortisation is computed.

Cost of Consumables		
Item	Price per unit	Total cost
Paper	0.05€	5€
Pens	1.97€	11.82€
Printer toner	32.98€	32.98€
Tin thread	9.6€	9.60€
Silicone bars	0.17€	1.70€
Total cost:		61.00€

Table 7.6: Caption

7.1.5 Indirect costs

This excerpt estimates the costs associated to indirect costs derived from the project activity, that is, the creation of software and assembly of hardware to produce the final product. Examples of these costs would be the ISP bill, electricity and water consumption costs and so on. We will assume these cost vary within the range of 15 to 20 percent of the cost of the project, so we will compute estimations using 17.5%.

Indirect Costs	
Total without indirect costs	10,254.62€
Indirect Costs	1,794.56€

Table 7.7: Indirect Costs

7.1.6 Total Costs

This section summarises the expenses of the project as a whole, as a sum of partial costs. Table 7.8 includes the price before and after considering taxes.

Total costs	
Human resources	9943.885€
Software Resources	159.02€
Equipment Costs	110.72€
Consumables	61.00€
Indirect Costs	1,794.56€
Total cost (No profit)	12069.27977€
Margin of profit (18%)	1,810.391966€
Total without taxes	14,199.15267€
End price including taxes (21%)	17,180.97473€

Table 7.8: Total cost

7.2 Socio-economic Impact

This section will discuss on the impact this project may have beyond the creation of the project itself, justifying its usefulness accounting for different environmental fields; that is, stating the repercussion of the work in different areas that may be somehow affected by it.

First of all, the system has a direct impact on the author's home studio itself, enabling him to use the gesture controller subsystem as an e-drum trigger to use in

conjunction with *Cubase* and *Addictive Drums 2* to add drums to his own songs while leveraging traditional drumming technique. It allows for an enhanced practice by including real-drums sounds as a consequence of physical hitting, which will be specially positive to develop limb coordination.

On another topic, this project provides a wide introduction to Virtual Musical Instruments, along with the technologies and approaches that have been used over the years for the design of these innovative ways of generating music. Consequently, Chapter 2 of the present document is likely to have an positive impact on readers keen on Computer Music and students looking for protocols that enable the transmission and storage of musical events within a computer system. It may be specially useful to gather resources in order to get started in NIME creation, as well as understanding the technical terminology surrounding these research fields. Plenty of research papers are included as bibliography and summarised in broad strokes so that information is easier to take in, which can help beginners to figure out the way to go for their projects to come to life, specially when it comes to analyse feasibility. Furthermore, the MIDI protocol is covered in-depth in this section, as it played a huge role in the dissertation and its basics are may be of interest to any music enthusiast.

On the other hand, the MIDI controller designed in this project is fairly easy to replicate and constitutes a quite challenging task that anyone interested in building a MIDI controller would find helpful. As source code is available to the public via GitHub, improvements to the algorithms presented here can be suggested by the community, or people can use the project as a base upon which they can build much more complex systems. It is also a good way to get started with time management and embedded systems programming, handling timers and problems derived of overflow in those. As a final comment on this aspect, note that a low-cost electronic drumset trigger can be built by following the process described previously (with a cost of around 110€at most in hardware) , making this project appealing to people interested in learning to play drums or expanding a home-studio while leveraging programming skills and creativity (as it is the case for the author of this document).

On another matter, the elicitation process performed and detailed in this dissertation can be used as a reference for future work, at least regarding the author himself. Revisiting the fundamentals of Software Engineering was appreciated and paved the way for the rest of the project to come to life, transforming a fuzzy idea into a clear objectives that could be finally approached. Context diagrams, Block diagrams and UML tools in general helped making the ideas in my mind more concrete and certainly made the process of explaining the idiosyncrasies of the system much simpler, aiding readers in understanding the content as well.

Finally, this project is conceived by the author as a starting point for further work and research on the Computer Music field and thus, there is a possibility that in the future, a PhD may be expanding on this project, including improvements and advanced functionality that could not be explored herein.

Chapter 8

Legal Framework

This chapter covers the regulations and licenses that may apply to this project, alongside the licences regarding third-party software used to provide integration of subsystems within this project.
First, Section 8.1 will ennumerate European regulations of major applicability to IT systems, discussing applicability to the project itself. Then, Section 8.2 will identify the licences involved in the system developed for this dissertation.

8.1 Legal concerns

Since new laws have been created in the recent past which are specially oriented to control data management in IT systems, that is, towards enforcing personal data protection. It is worth mentioning them briefly, stating whether they apply to the system depicted in this document.
To begin with, at the European level, the General Data Protection Regulation (**GDPR**) entered into force in May 2018 (read at [81]). This Directive has direct applicability in Spain and aims to make legislation homogeneous across the EU (European Union), providing users with more control over the data they share with all kinds of IT companies. Thus, the directive is applicable to data controllers and data processors, that is, companies or organisations which tell the purpose of data gathering and processing in the first place, and which later perform automated or non-automated data processing.
In case of handling PII (Personal Identifiable Information), a company shall evaluate the data and enforce privacy, as well as guaranteeing basic rights to the users.
This European regulation sprouted the creation of a new Spanish law, abbreviated as **LOPDPGDD** (Ley Orgánica de Protección de Datos y garantía de los Derechos Digitales) that further constraints the European Regulation. Amongst the main articles of interest regarding this law we find user rights regarding PII, summarised next:

- **Right to Access (Art. 13):** a person shall be able to retrieve the information a data processor or controller keeps.

- **Right of Rectification (Art. 14):** a person shall have the ability to modify any incorrect data kept by the data processor or controller.
- **Right of Elimination(Art. 15):** a person shall be given the ability to have their personal data removed from processing and control. However, the data controller may keep the required data in order to avoid direct marketing.
- **Right to limit data treatment(Art. 16):** a person shall be given the ability to limit how data is treated in the following circumstances:
  - The user refutes the accuracy of the kept data.
  - The data controller entity holds and processes data illicitly.
  - The user does not want the data controller to treat their data but wants it to keep data in order for the user to make a complaint.
  - While the the right of opposition has been exercised and its application is under verification.
- **Right of portability (Art.17.):** a person shall be given the ability to access their own PII and transmit them to a different entity responsible for them from that point forward.
- **Right of Opposition (Art. 18):** a person shall be given the ability to refuse to receive direct marketing and the ability to oppose to data processing unless a special imperative situation arises (e.g. a public interest).

The whole document is accessible at BOE (Boletín Oficial del Estado) webpage [82].
Some interesting articles that expand on the European regulation are Article 7, which provides special coverage for minors (whose consent to data processing will be valid only once they are 14 or older) or Article 9, which states the categories of data that cannot be handled (e.g. those that have as main target to identify ideological aspects, religion, sexual orientation, etc.) unless special circumstances take place.

Since our system does not handle any kind of personal data by the end of this dissertation, the aforementioned regulations do not apply to it. The author will be specially conscious in case the system eventually processes or gathers personal data, so that the system complies with the regulations on this fundamental aspect of privacy nowadays, being aware of the different levels of infractions, the consequences of data leakage and derived legal implications in case of infringement of the law terms.

8.2 Applicable Licensing

The system developed bases integration upon software developed by other people; the main programs used in this work are Hairless MIDI, loop MIDI and the MIDI Jack Unity Add-on.

Hairless MIDI is distributed under the GNU Lesser General Public License 2.1., which is a permissive version of the GNU GPL (General Public Licence) Licence, which allows the use of a Program [1] within a non-freely available work. That means that anyone could create a given software product including the GNU LGPL licenced piece of code and redistribute the result as privative software unless the new piece of software is a derivative work (i.e. a work that is based upon the LGPL software and modifies it would be derivative, whereas simply leveraging its features would not be considered as derivative work).
Therefore, source code from Hairless MIDI could be modified in the future and published using either GNU GPL or LGPL to, for instance, include the core functionality of it within the source code of the Unity-based videogame presented in this dissertation.

MIDI Jack is simply copyrighted, according to the GitHub page where you can find the source code for the whole project; anyone is allowed to convey copies of the software without restriction, being allowed to use, copy, modify, merge, publish, distribute, sub-license or sell those copies but keeping the copyright notice found in Keijiro's page in all copies or substantial portion of Software.
On the other hand, LoopMIDI is a privative, free-of-charge program copyrighted by Tobias Erichsen, which can be used for private, non-commercial use. In the case commercial use is intended, the author of this program must be contacted.

Lastly, regarding other licences, the Arduino environment is covered by the GPL as well, which allows the use of the libraries without restrictions on commercial and non-commercial products.
Only modifications to these core software are required to be freely available. Regarding Arduino boards hardware, this is said to be Open source, meaning that anyone should be able to study the hardware, make changes to it and share changes. Thus, files associated to the hardware (Eagle CAD files) are released under a Creative Commons Attribution Share-Alike license (Arduino [2], which requires anyone modifying the hardware to notify so in the modified files and distribute contributions under the same licence.

[1] Program: a work that is released under a GNU Licence
[2] see details at https://www.arduino.cc/en/Main/FAQ

Chapter 9

Conclusions

This chapter reflects on the results obtained from the project, discussing the objectives that were achieved after months of work, providing with proposals for future improvements that may be carried out and personal comments that will examine the greatest difficulties found along the project. Section 9.1 will analyse to which degree the objectives initially proposed were attained. Section 9.2 will deliberate the future work that is aimed to expand the capabilities of the system in interesting ways. Section 9.3 will contain personal conclusions regarding the experience of development, in terms of technical expertise gained as well as acquired life lessons.

9.1 Project Retrospective

At the very beginning of this document (see Section 1.2), a set of objectives were enumerated, accounting for the system to be built. This document covered the whole process for a electronic musical instrument to come to life, including both a tangible interface for interaction as well as sound generation subsystem, which although they are fairly restricted in functionality, have paved the way for further research on this topic.
Next, the set of objectives will be listed along with comments support their fulfilment.

- **Objective 1: Creation of a physical interface for musical generation**: a tangible interface involving drumsticks with embedded electronics was created, along with a set-up involving a custom hi-hat pedal based on an LDR sensor.
- **Objective 2: Creating a 3D visualisation application that represents the drumset in virtual space and enables triggering sounds associated to the different present components:** an application based on the Unity game engine was created in this project, exploring the virtues of this tool for development and making use of previous work to avoid reinventing the wheel while learning a lot.

Secondary objectives such as real-time concerns, usability or aesthetics were also implemented to the extent possible, since time and budget limitations existed. On this matter, the system is responsive enough to provide a natural performance to the user, even though improvements would play a huge role in playfulness and user experience.
Regarding aesthetics and ergonomics, an extra effort was made to make User Interfaces of the software application appealing, as well as making the hardware counterpart solid, neat and seamless to use, trying hard to keep distractions away from the users, so that they can focus on the experience.

On top of that, this project has involved a plenty of research, which is manifest in the State of The Art excerpt devoted to previously developed musical instruments (see Section 2.3); providing readers with a exceptional interest in the Computer music field with a quite detailed summary of sundry research papers referenced along the dissertation.

On the other hand, it is worth mentioning the fact that this project has changed a lot in essence since it was first conceived in 2017, as the author progressed through the Informatics Engineering degree. It all came to mind when the author got to know Microsoft Hololens and thought it would be a would idea to apply the Augmented Reality technology to one of his passions, music. As space had always been a problem for him, he envisioned a system which could project a complete drumset in front the eyes of the users to allow them play drums ubiquitously, but hitting real surfaces, like say, the user's bed (which was at that point used as practice drumset during breaks from studying by the author himself).
Consequently, and inspired by the opportunity given by the supervisor of this dissertation (which was happy to let the author use an actual pair of Hololens), he targeted at eventually, giving birth to such system as a Bachelor Thesis.
Unfortunately, the reality is that at the end, the original concept solution turned to be too complex and certainly unfeasible to be implemented in a timely schedule due to the huge amount of expertise and effort required to complete effectively such self-built expectations. Unsurprisingly, working with new technologies which were not very well documented (as they are in their infancy) and doing so alone imposed hard constraints in the outcome of the project, which had to focus on providing an intermediate solution, going away from the AR paradigm (at least giving up Hololens support) without deviating completely from the main objectives of the system and aiming to leave room for improvement so that the initial concept can be ultimately achieved.

9.2 Future Work

The development of this system has been carried out in two phases or builds, which progressively added features. Regarding the limited duration of the project and the constraints associated to working on several subjects concurrently, many additional

features were looked into yet left out because of the limited time and monetary resources inherent to the dissertation deadlines and the low purchasing power of the author. Thus, work in progress and future directions for the expansion of this work will be listed and discussed next:

- **Orientation Detection support**: one of the major lacks of this project is the limited number of sounds that can be obtained from interaction with the tangible interface presented in this document. The original idea involved triggering different sounds depending on the orientation of the drumsticks in the real world, as well as representation of the drumsticks in virtual space. This subject has been widely explored and was planned to be included in this paper. Unfortunately, unexpected problems arised and the addition of this feature could not be completed nor documented in a timely manner. The sensors which are being used to support this functionality are MPU9250 IMUs(Inertial Measurement Units). The author is developing a library to deal with this sensor, based on its instructions sheet and with aid of similar work, such as borderfligth's Github project. By creating his own library, the author is striving to learn basic hardware concepts that can help him deal with devices which have no libraries specially supporting their easy integration. Understanding IMUs and calculations required to compute orientation are thought to be very interesting due to its applicability in many research areas.
- **Explore MIDI BLE**: as the system is currently limited in flexibility and ergonomics due to the wiring required to interconnect components, one idea for the future is to explore the the inclusion of Bluetooth Low energy modules to Arduino. Other Arduino-compatible boards may be considered, taking into account the possibility of embedding a small microcontroller with BLE support within drumsticks, as in the case of Airstic drum.
- **Support for Hololens**: the ultimate objective of this project is AR visualization using a see-thorugh device such as Microsoft Hololens. Exploration of this support to date from the perspective of the author has been unsatisfactory. However, further research will be performed to assess feasibility of this highly-desirable feature.
- **Support interruptions in the gesture controller side in order to guarantee that user hits are never missed**: hardware interruptions allow to increase the amount of simulated multi-tasking the microcontroller supports by associating function calls to specific hardware events (such as pushing a button). This opens up many possibilities for improvement of the subsystem.
- **Add a UI to gestural interface**: in order to provide the user with verbose information about the status of the subsystem, an LCD could be used to help the user understand potential problems. It would also help debugging the system and could be used to implement configuration during runtime, by adding rotary encoders or other sensors.
- **Support air-drumming as well as hi-based interaction**: similarly to Airstic drum [42], both interaction modes can be enabled to further increase the flexibility of the system. Air drumming could be used to trigger cymbal sounds whereas drum sounds may be triggered by means of hitting.

- **Add animation to virtual objects on interaction**: in addition to simply changing the colour of drum pieces when the user interacts with the system, realistic movement of the cymbals can be included to enhance the appeal of the application as a whole.

9.3 Personal Conclusions

After more than ten months of work and dedication, this document and the additional effort required by the actual development of the system described herein have all finally come to an end.
This makes the author incredibly glad and proud of the achieved result, even though the system differs quite a lot from the initially devised concept.
The author's humble home studio has henceforth a new instrument to be used in musical projects of his own, which is probably the main source of satisfaction out of the whole project.

Countless hours and will power have been put into making the best out of this project, taking care of the aesthetics and most importantly, the quality of the content and explanations. All of this attention to detail makes the author strongly believe that each and every page of this dissertation is worth reading.
In this sense, it can be safely stated that his project has constituted the most difficult challenge the author has had to face in his entire life, both in personal and professional aspects:
On the one hand, the author has been required to complete this project while working part-time on a different subject to help the family income, which confined the project to progress much more slowly than it could have otherwise. On this matter, developing ***DrumVR*** has been specially tough when dealing with the Augmented Reality paradigm on the software side, since the early objective of developing for Hololens progressively seemed more and more unfeasible due to the difficult access to the Hololens (as it took to the author around 2 and a half hours to get to University using public transport). Testing on the Hololens Emulator was simply too overloading for the developer's equipment .
Furthermore, this development, research and documentation process has required extra time resources due to the low level of expertise the author of this paper had previous to proposing the idea to his tutor. Being honest, the author had not any prior knowledge about the technologies used for this project beyond some Arduino programming basics along with some 3D modelling skills. The hardware selection has been probably one of the most uphill processes due to the obliviousness of the author regarding this topic, prices, types of sensors and compatibility of those; electronics were all gone from the author's mind when this project started.
On top of that, this document itself has been fully written using LaTeX , which has been incredibly useful and work-saving regarding citations, cross referencing, page numbering and sectioning of the document, as well as basic styling. However, sometimes, it has made the process of writing much more tiresome that it would have been if the author had used a common text processor such as Microsoft Word,

becoming a distraction from the content when things did not work as expected. This was the case when creating tables and styling those, as well as when dealing with large figures, a topic that was hard to deal specially for traceability matrices. Regardless of the additional effort derived from using this LaTeX the author has been able to learn a lot about it and can now estimate the kind of processes that can be speed up using this tool and which are better off with other text processing software.

More personally speaking, this project has reasserted the fact that continued effort is tiresome yet gratifying, and has served as an educative experience to get to know my limitations and strengths better; understanding the importance of mental health, specially, for a job such as those related to the IT field, requiring a lot of work and often, isolation and time for introspective. In this sense, the author has discovered a research path of his interest and has saw the doubt on future decisions regarding his career, finally inspiring him to apply for a Master's Degree oriented to research.

On another matter, the uncertainty surrounding the project and the difficulties regarding its feasibility have made the author realise the amount of knowledge that is required to estimate resources required for the completion of a system accurately. This is an incredibly complex task whose extent is really difficult to encompass alone, specially in the case of projects targeted at new platforms; it is a great responsibility and takes a huge expertise.

Owing to the numerous difficulties that have been found along the way, the author has had the opportunity to appreciate his mistakes as a necessary step for success;in light to that, it is important to give credit to one's efforts regardless of the visible results in order to keep one motivated. If nobody values the process required to make things happen, you are the one that should do so because otherwise, there is no point in trying and you will never achieve a thing.

This project has certainly made the author appreciate clear guides to learning, such as those provided by institutions like Universities, in the shape of notes, presentations or class projects. These are invaluable and make learning so much easier than simply finding your way out of problems without a clue about where to find a solution.

All that been said, the author is looking forward to continue expanding the system depicted herein, which eventually, will become much more useful and exciting to showcase. It has been a great opportunity to merge two passions, music and programming, expanding the author's knowledge and testing out his capabilities through adversity; but most importantly, has constituted an experience of personal growth.

Appendix A

Side Notes

One important aspect of Virtual reality applications is presence, which plays an important role in the success and usability and enjoyment of them. One way to aid presence, also known as immersion, is to provide high quality graphics (High Reproduction Fidelity), or at least a level of graphics that best fits the central purpose of the application. In the case of the presented system, a great effort was put onto the refinement of the graphics of the virtual counterpart of the drum-set to be played, enough so that the user can feel somehow interaction is similar to that of playing a real instrument in a sense.
Since the project was initially targeted towards an AR application (to be used with an HMD), the 3D models generated for the project were carefully modelled with real-world sizes; using multiple references for modelling from drum hardware brands such as Tama or Pearl, and cymbal brands like Zildjan or Sabian.

In this appendix, a description of some of the most cumbersome processes found along the dissertation can be found.

A.1 How to configure Blender to work with real world units

The firts thing that has to be done before modelling in real-world units is configuration in Blender, which is the modelling program used all along this project.
Next, in order to ease the process to any newbie to the subject, I will describe the process to get started with modelling in meters. Note that these instructions may not be as accurate for versions distinct from 2.79, the one used by the author of this paper.

1. Access the **Scene panel**  within **Properties** view 

in any Blender area of your choice (that is, any subdivision of the main window).
2. Go to the **Units** submenu and set the Length property to **"Metric"**. You

should be able to see the word "Meters" under the Type of View Displaying (Ortho or Perspective).

REMARK: ***Perspective view*** mimics reality in the sense that perspective distortion is present, that is, relative sizes are hard to tell for two objects that are apart from each other. On the contrary, ***Orthographic view*** is unreal, since no distortion with the distance is present; i.e. if two lines are parallel and we look at them, they look like they are actually parallel (they never cross), which doesn't happen in the real world (where they seem to cross in the horizon).

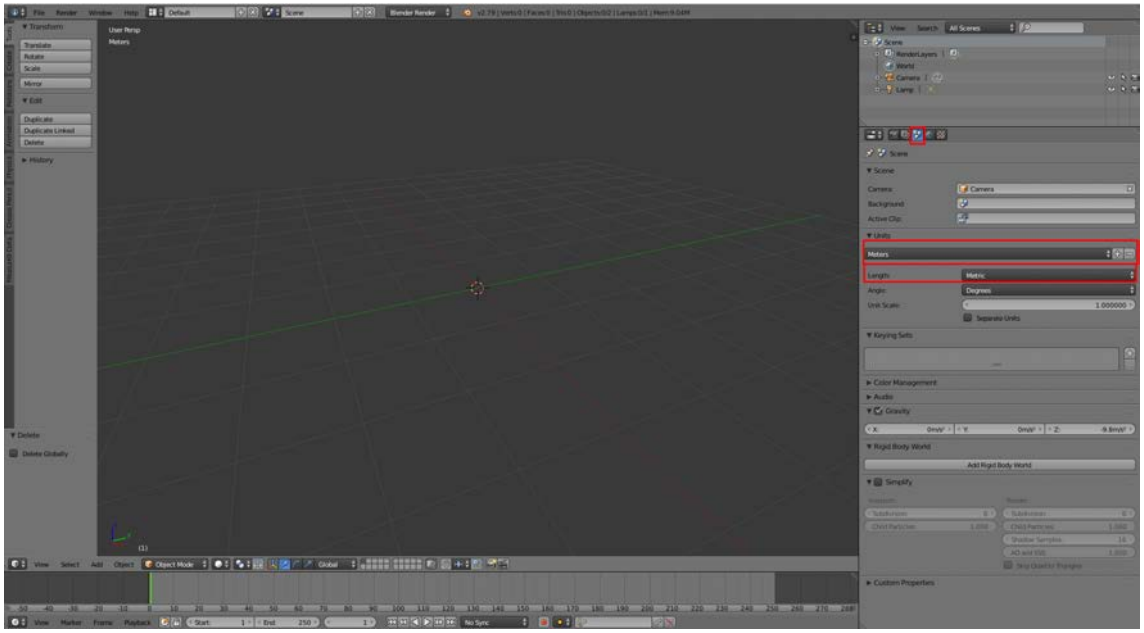
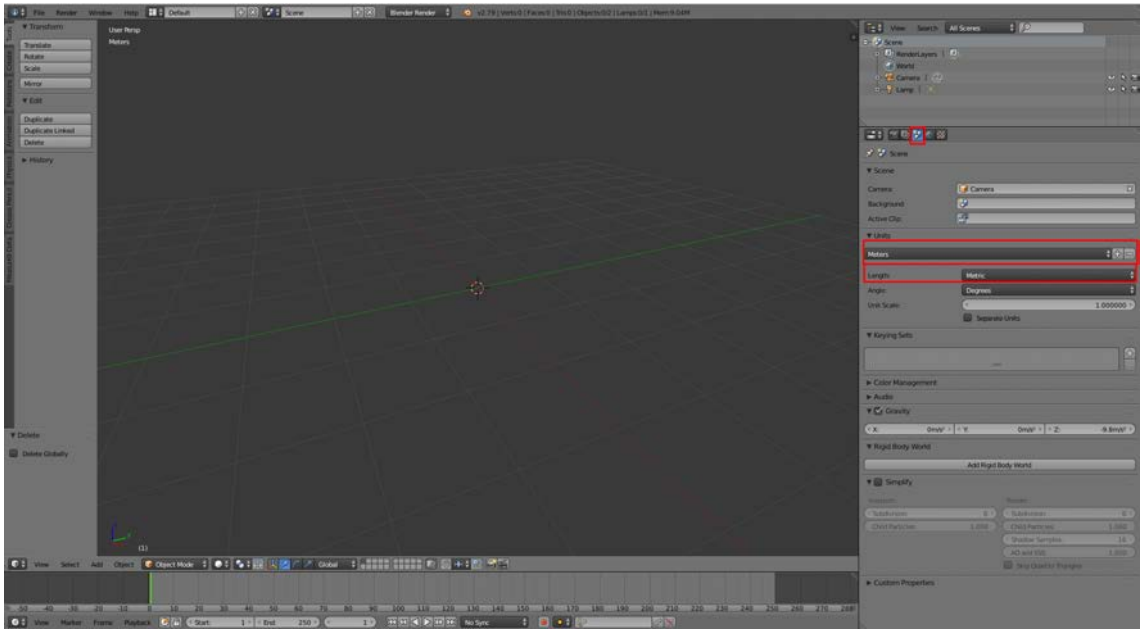
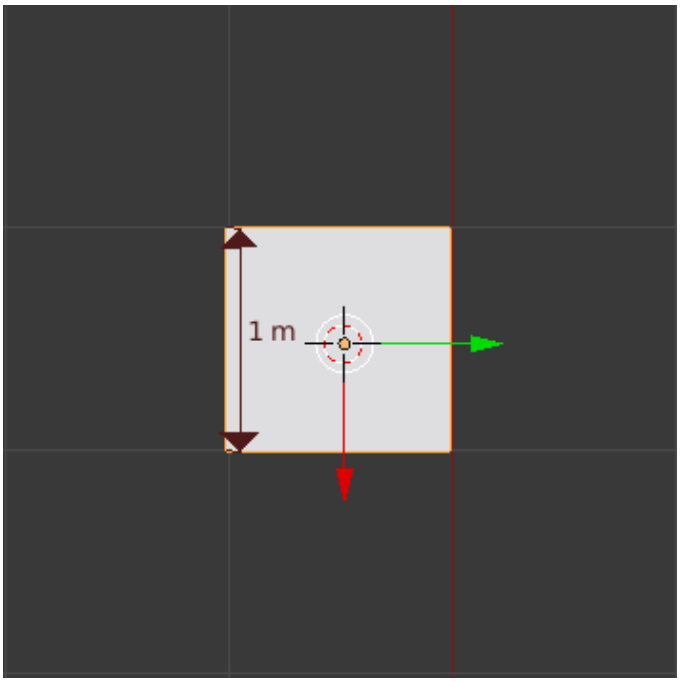


Figure A.1: Metric Submenu

* Then, you may want to specify the kind of unit you are more comfortable with, in order to work with angles (either degrees or radians). You can change change that next to the **"Angles"** label (see Figure A.2a).
* Additionally you may want to set the **Unit Scale** down so that the grid Blender provides in the ZY plane is visible again, knowing that a Unit Scale of 1 means that if we input a value with no units, they will be assumed to be meters. Therefore, that value can be changed to state all your measures in millimetres by setting such value to 0.001.



(a) setMetric



(b) Resulting Blender grid

Figure A.2: Blender 3D Grid for a Unit Scale of 0.001

Open Source Free Software has been mostly used for the modelling process, as it is the case of Blender. Addons like *MeasureIt* were found to be extremely useful for modelling with real world sizes real hardware was used as reference to obtain an enhanced precision towards the final user of the system, which is also the developer. It can be enabled directly from the Blender User Configuration menu, and it is very simple to use. In figure A.4 you can find a render using the aforementioned Add-on, in order to visually keep track of the dimensions of the different drum-set parts while building the scene.

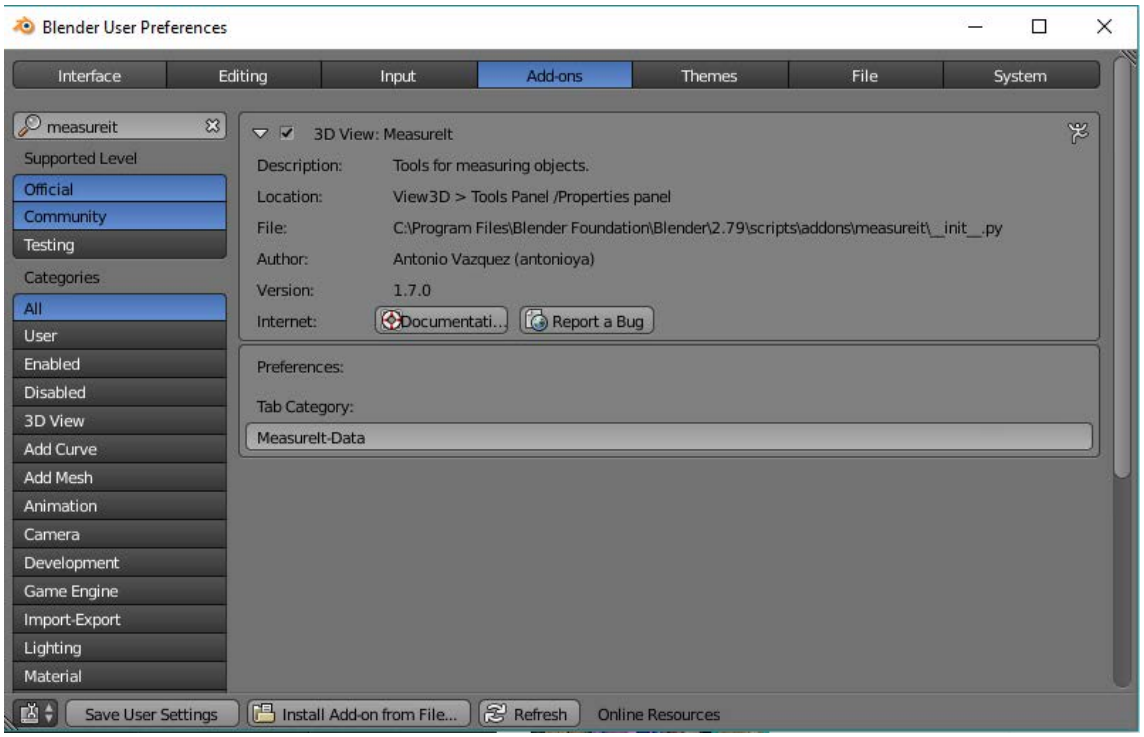


Figure A.3: MeasureItAddon

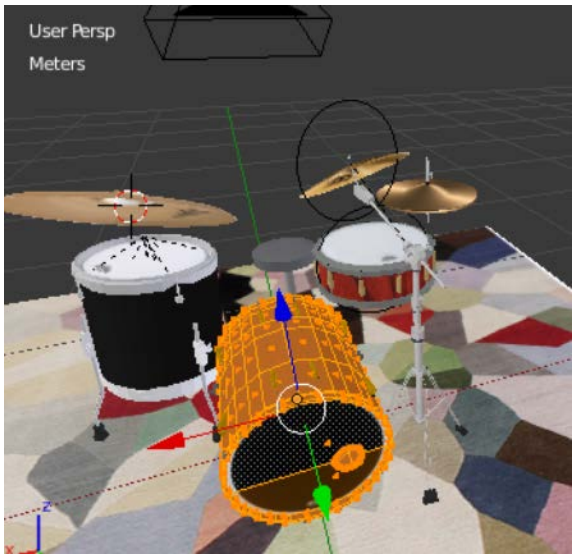
Figure A.4: Final render using MeasureIt Blender addon

A.2 Baking for Unity

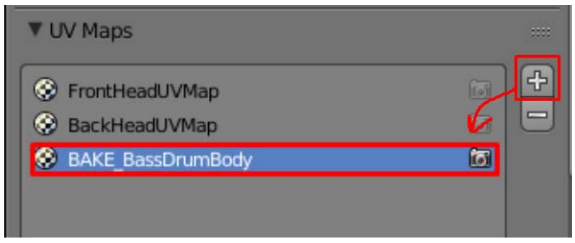
According to many 3D artists and videogame creators I have followed (e.g. Widhi Muttaqien, Andrew Price [aka Blender Guru] or Asbjørn Thirslund [aka Brackeys])

for reference during the development of this project; game assets are to be created using the **Blender Render** render engine. The reason for that is precisely compatibility, since materials work pretty much the same they do in game engines such as Unreal or Unity. However, the author realised about this workflow after he had already completed the detailed 3D models using Cycles render.
As Unity and Blender cycles textures are not directly compatible because a game engine can't deal with continuously updating textures on all objects of a scene due to changes in light; textures need to be **baked** so that they can be used in Unity.
Baking is a process that consists in taking a snapshot of how textures in a scene look under certain lighting conditions so that we can leverage the beauties of PBR while having real time interaction. In other words, the result of a render given the parameters of the current scene, is saved into an image file (used as image texture) for latter use.
Baking is useful when we have fixed lighting in a scene, as we can simply paste a texture looking like it was rendering in real time but instead we pre-rendered how it looks. Baking can be done for both simple image textures as well as procedural textures and is specially useful for animation and exporting to different software.
The baking process is somewhat hard to understand at first, since you need to deal with UV maps, a delicate topic for beginners.
However, I will cover next the process involved into baking the textures for my whole scene, aiming to make everithing easy to understand and follow.

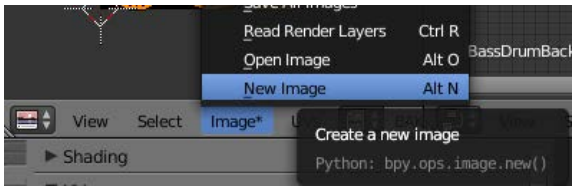
1. Select the object whose textures you aim to bake onto a single one (e.g. the bass drum mesh).



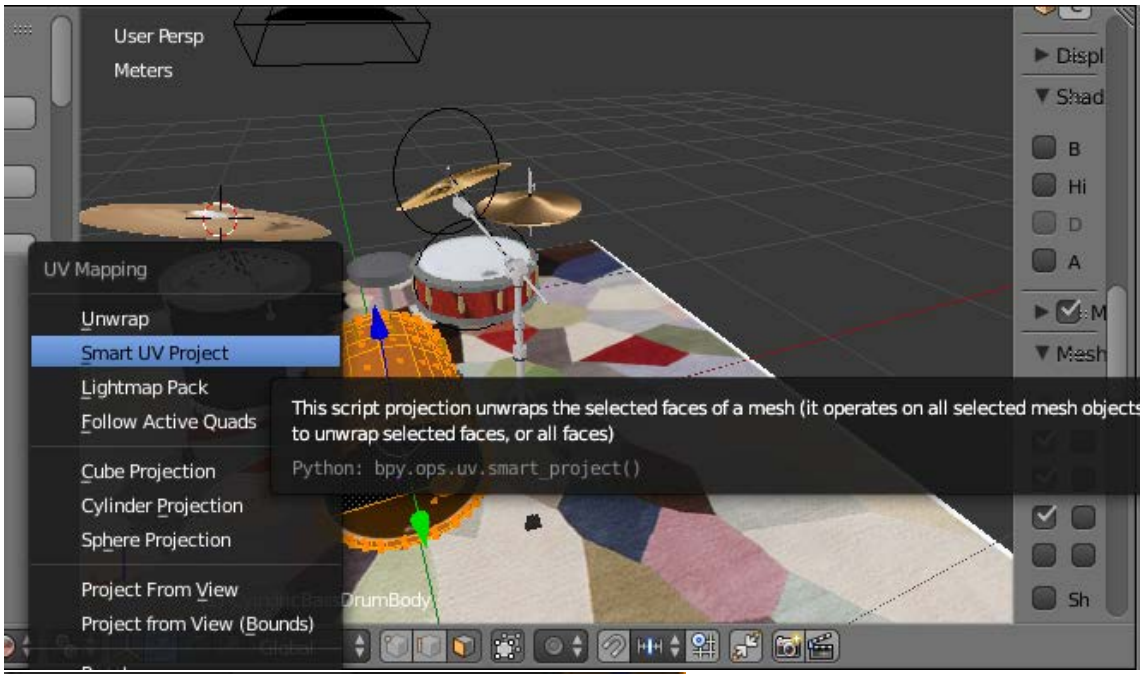
2. Create a new UV Map, give it a name of your choosing:

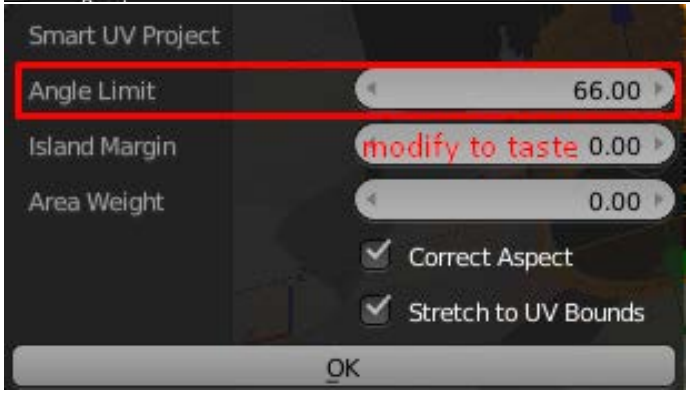


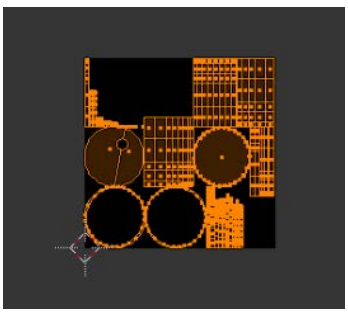
3. Create a new image for the Bake output, the texture to be exported. In order to do so, we open the Image/UV view and Go to **Image → New Image** and set a the dimensions of the texture we would like to create. 1024 px is fine for both width and height unless you are going for a very high-quality texture bake or your object is really huge and details do matter.



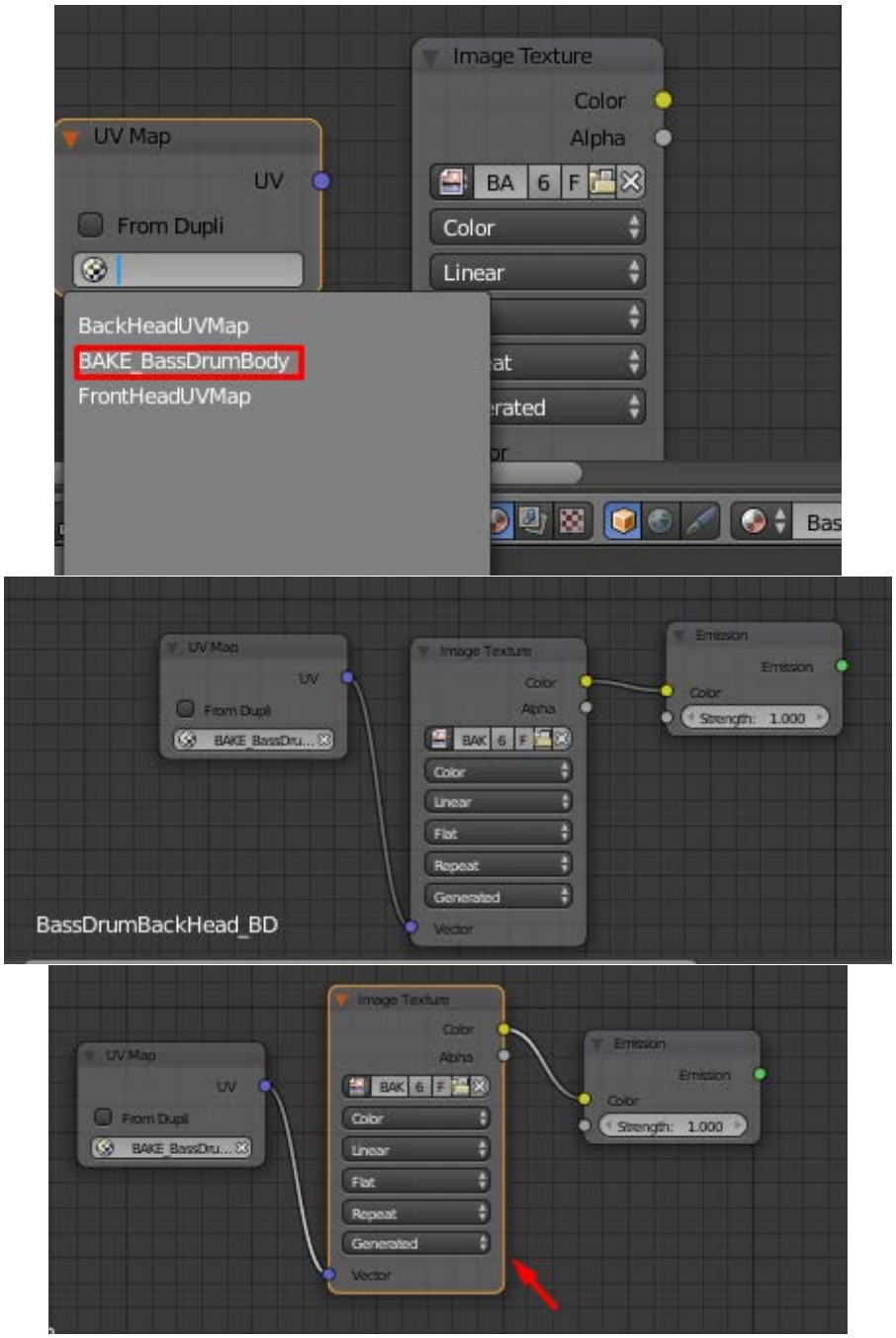
4. Go into edit mode with the target mesh selected and perform **Ctrl+A** to select all the mesh. If we have created seams for unwrapping our mesh very precisely, we can directly mouse over the 3D viewport and press U to display the UV Unwrapping options. UV unwrapping is a technique to project a 2D texture onto a 3D model, and it can be thought as unfolding our model so that all its faces lie flat and can be rebuilt creating a smooth texture. For simple models, the Smart Unwrap option will work well; since you can customise the angle that is used to compute seams (texture cuts) automatically. Otherwise, you must define your own seams so that the Unwrapping doesn't look like a mess.



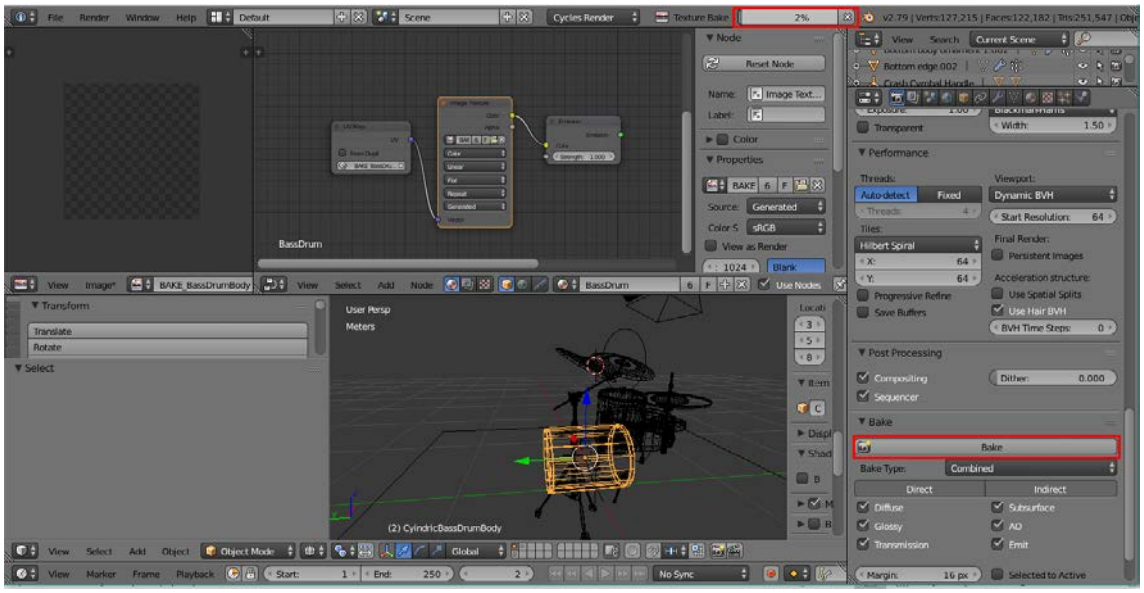




5. Once UV unwrapped, the UV/Image editor should be showing the unwrapped faces all over the black texture map we created in step 2. Now, it is time for us to tell Blender to bake the textures assigned to our object onto the image we want. To do so, we need to create a set of nodes for each material that is present in the mesh whose materials we want to bake. For each material, we need to create at least One UV Map node (to reference the UV map specifically created for the baking process), one Image Texture node (to tell Blender where to bake) and an Emission node (in order to preview the results of Baking).
6. Set the Texture Image node source to be the Image we created previously in all the Image Texture nodes we created in the previous step. Then, make sure you left-click on the Image Texture node in all the materials you have applied to your mesh (you can see if they are selected by looking at the orange border they should show).



7. Make sure you have selected the Image Texture Node (it shows an orange outline). Go to the Render Panel, within the Properties view. Find the Render Submenu and Click **Bake**.



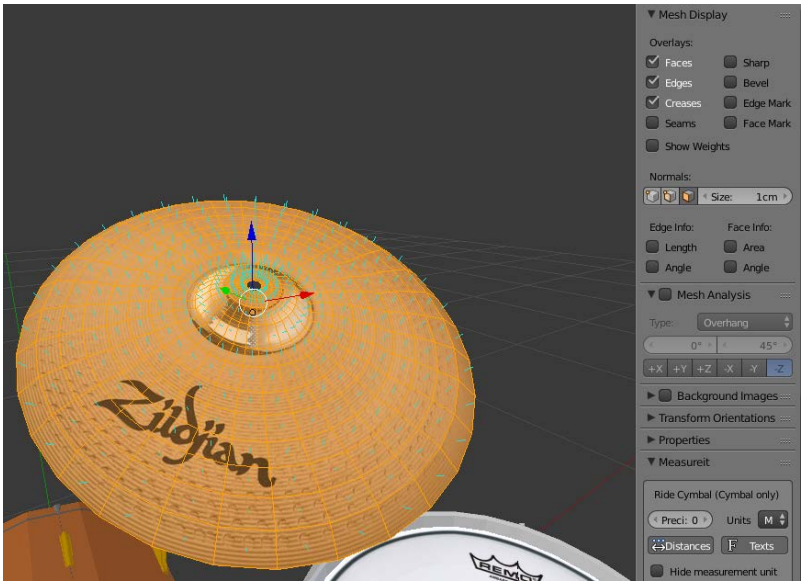
8. Check whether you like the result or there was some kind of issue.

Figure A.7: Render of Modeled Drumset after Baking

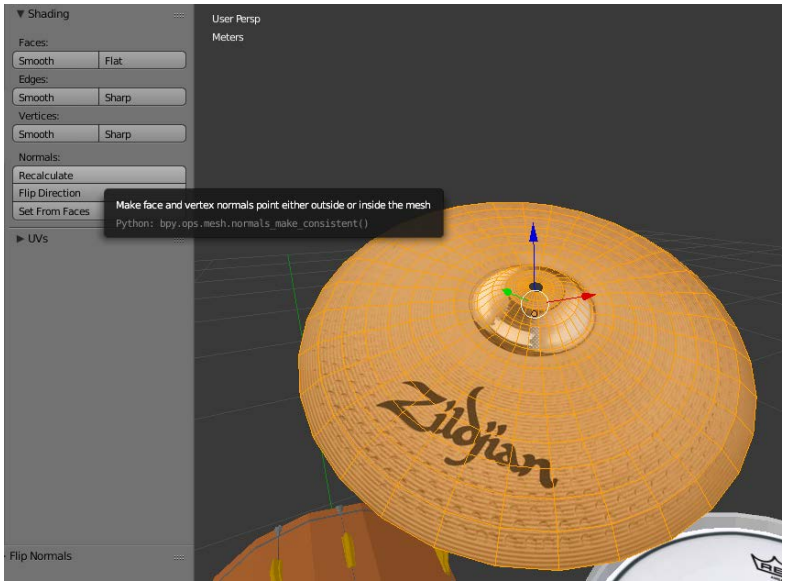
A.2.1 Bake Troubleshooting

A completely black render is a clear sign that there was an issue during the process. The first thing to double check is your normals. There is a strong possibility that the normals of your model are pointing inwards, thus, messing up your render, since no light gets into the model (obviously)
In order to recalculate the normals of a mesh, you can Go into Edit Mode ¿ Press T (to open the Tools menu) Go into Shading/Uvs and press Recalculate (within the Normals Submenu; see Figure A.8b).
In order to visualise the direction of the normal vectors, you can go into Edit mode, Press N to show an auxiliar panel and Find the Normals Submenu. There you can

choose among showing vertex normals, edge normals or face normals along with their size (see Figure A.8a).



(a) ShowingNormals



(b) Recalculating Normals

Figure A.8: Baking Issues: Fixing Normals

Another possibility is that you are using a glossy material, which cannot be baked. In this case, you will have to change it to a Diffuse one or any other compatible Node.

To end this appendix, we show the result from all the bakes of the final scene, using within the Unity game Engine for development. Figure A.9 shows the images obtained from the baking process.These are the ones pasted into the models themselves to avoid real-time calculations.

Figure A.9: Image textures for the whole drumset

A.3 Panoramic view of the development process

This last appendix provides with a set of figures showing progressive refinement of the system described along this document. It has been added for completeness and so that the reader can get a general idea of how the depictions of the system provided along the document map to the actual thing.

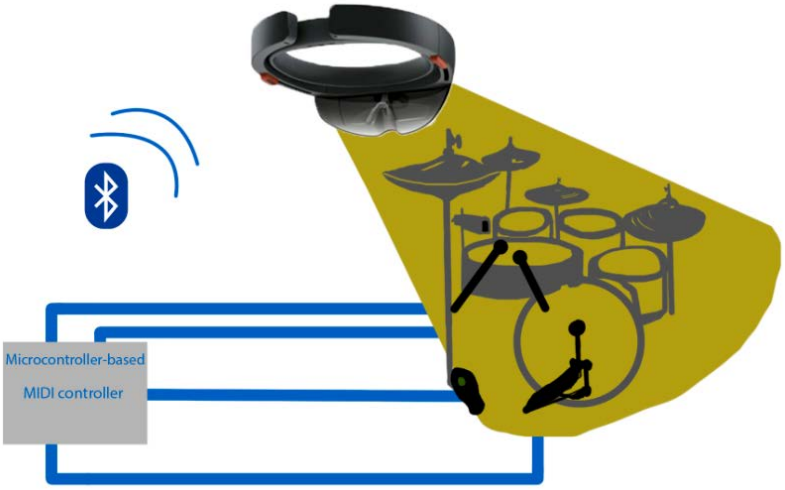


Figure A.10: Initial Idea Draft

Figure A.10 shows the envisioned system at the first stage, reflecting in broad strokes the market product that was conceived initially. It is manifest that the idea suffered multiple adjustments in order to be feasible in a limited budget and schedule.

At the very beginning of the project, on the software side, a Piano-like video-game was created, used as a basic project to learn about Unity. It was used in conjunction with the MicroKeyboardByNolliejandro MIDI controller (see Figure 2.16) to trigger sample sounds, and it allowed to generate piano tiles procedurally.
Notes were automatically assigned correctly as the piano grew in number of notes and sounds were configurable. Similarly to the final solution for the drumset; keys turned red as MIDI input was received corresponding to specific notes.

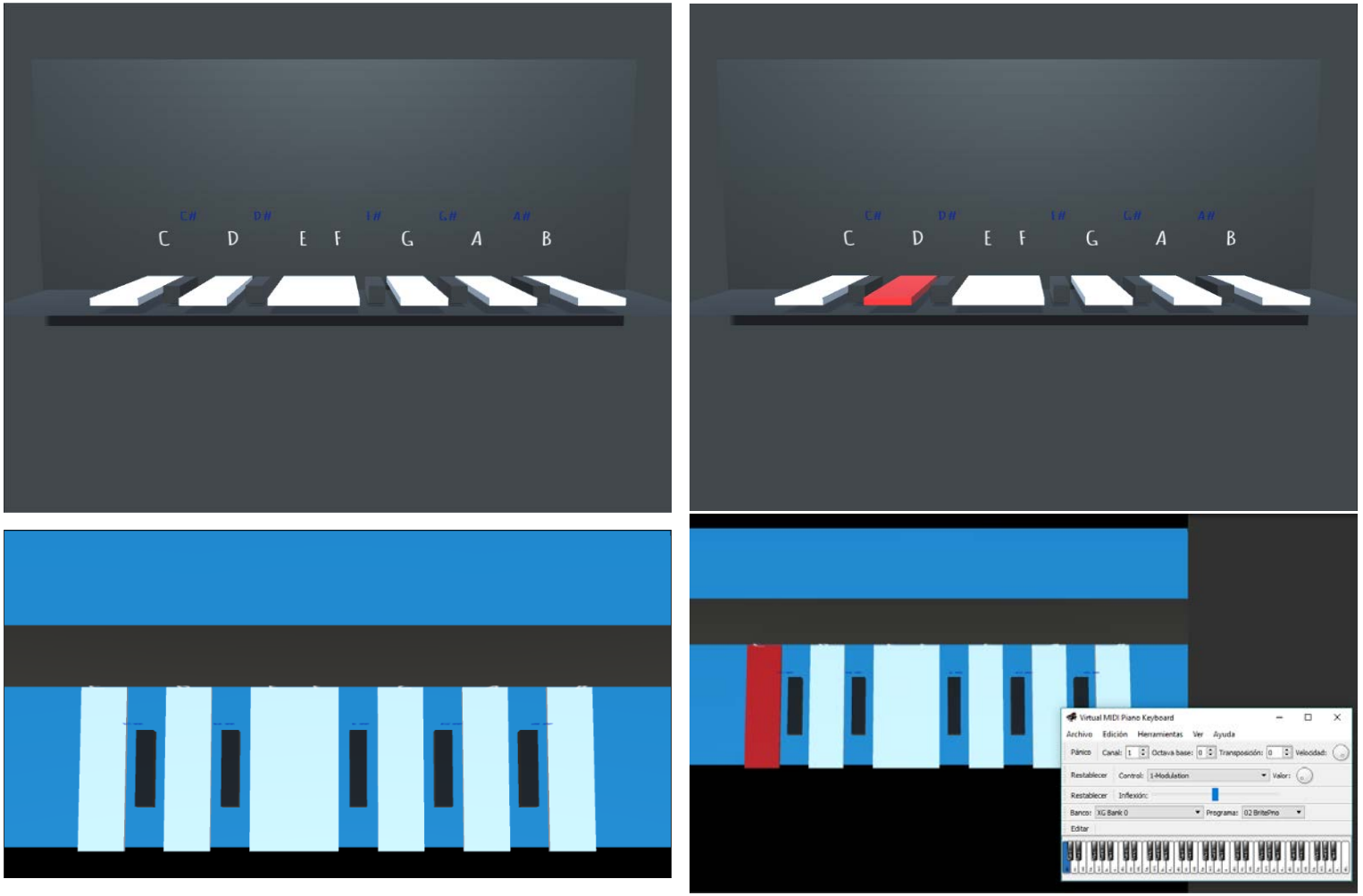


Figure A.11: Initial Unity Project

Figure A.12 shows one of the earliest versions of the MIDI controller subsystem, when it did not involved USB wiring, nor included soldered joints of any kind. In the picture, we can observe

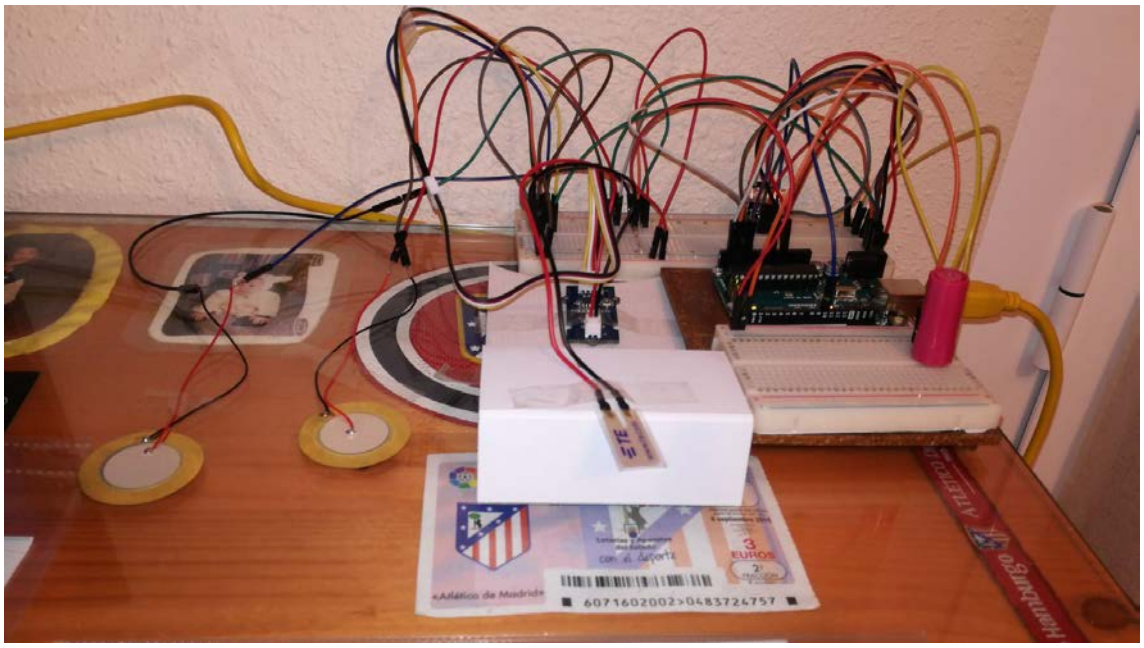


Figure A.12: Early version of MIDI Controller

Figure A.13 shows the main components of the tangible interface created in this project, embedding hit sensors (in the case f drumsticks) and a LDR in the case of the left foot pedal. These are connected to the main Arduino board using a male-to-male USB cable; having a female soldered end in both the drumsticks/pedal and the Arduino wiring. An example of soldered joints regarding USB females can be observed in Figure A.14.

In Figure A.15 we can observe all of the 3D models created for this project, along with their textures. These aimed to provide high Reproduction fidelity to give realism and appeal to the application, which otherwise would have look unprofessional.

Figure A.13: Tangible interface without USB wiring

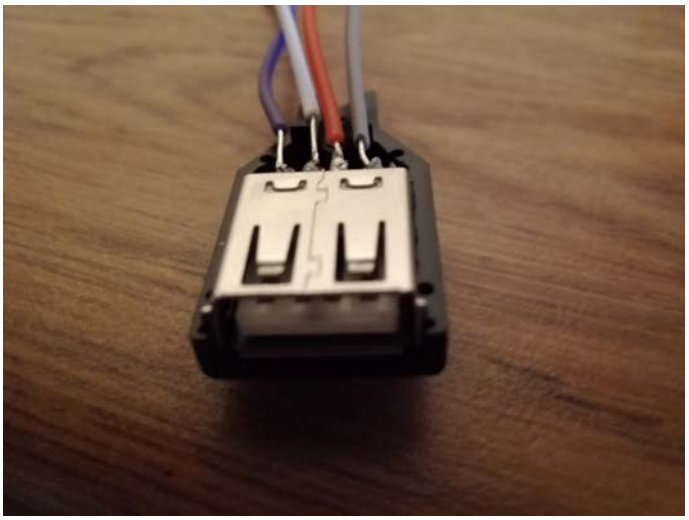


Figure A.14: USB soldering

Figure A.15: Piece-by-piece drumset renders

As mentioned in the Conclusions, some future work discussed was already being implemented, we can find evidence of that in Figure A.16, where a LCD screen has been added to provide the user with more understandable information regarding the status of the MIDI controller.

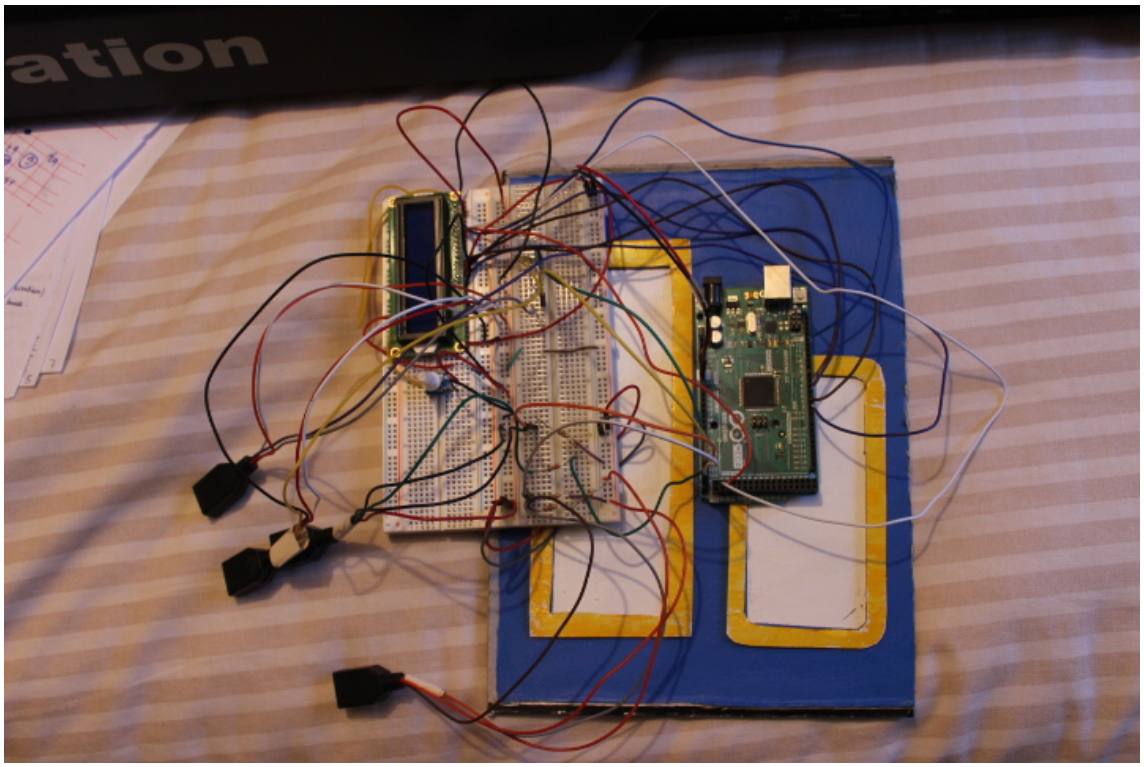


Figure A.16: Work in progress

Bibliography

[1] X. audio. (2019). Addictive drums 2, [Online]. Available: https://www.xlnaudio.com/products/addictive_drums_2 (visited on 06/09/2019).

[2] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino, "Augmented reality: A class of displays on the reality-virtuality continuum," vol. 2351, Jan. 1994.

[3] Wikipedia. (2019). Application layer, [Online]. Available: https://en.wikipedia.org/wiki/Application_layer (visited on 06/09/2019).

[4] ——, (2019). Tuple, [Online]. Available: https://en.wikipedia.org/wiki/Tuple (visited on 06/09/2019).

[5] U. F. D. R. G. do Sul. (2019). Nime, [Online]. Available: https://www.ufrgs.br/nime2019/e (visited on 06/09/2019).

[6] Wikipedia. (2019). Daw, [Online]. Available: https://en.wikipedia.org/wiki/Digital_audio_workstation (visited on 06/09/2019).

[7] ——, (2019). Virtual studio technology (vst), [Online]. Available: https://es.wikipedia.org/wiki/Virtual_Studio_Technology (visited on 06/09/2019).

[8] ——, (2019). Tangible user interface, [Online]. Available: https://en.wikipedia.org/wiki/Tangible_user_interface (visited on 06/09/2019).

[9] ——, (2019). Proprioception, [Online]. Available: https://en.wikipedia.org/wiki/Proprioception (visited on 06/09/2019).

[10] *Clavecin electrique, 1759*, Aug. 2018. [Online]. Available: http://120years.net/clavecin-electrique-1759/.

[11] *Musical telegraph, 1876*, Aug. 2018. [Online]. Available: http://120years.net/the-musical-telegraphelisha-greyusa1876/.

[12] *Telharmonium, 1897*, Aug. 2018. [Online]. Available: http://120years.net/the-telharmonium-thaddeus-cahill-usa-1897/.

[13] *Novachord, hammond 1939*, Aug. 2018. [Online]. Available: http://120years.net/the-novachordl-hammond-c-n-williamsusa1939/.

[14] *Sackbut synthesizer image*, Aug. 2018. [Online]. Available: https://ingeniumcanada.org/scitech/collection-research/artifact-hugh-le-caine-electronic-sackbut-synthesizer.php.

[15] *Sparkfun midi tutorial*, Aug. 2018. [Online]. Available: https://learn.sparkfun.com/tutorials/midi-tutorial/all.

[16] Dannenberg and R. B, "The interpretation of midi velocity," Nov. 2006.

[17] T. M. Association. (2018). Control change messages (data bytes), [Online]. Available: https://www.midi.org/specifications-old/item/table-3-control-change-messages-data-bytes-2 (visited on 01/03/2019).

[18] A. Rey. (2019). Microkeyboardbynolliejandro, [Online]. Available: https://github.com/coredamnwork/MicroKeyboardByNolliejandro (visited on 06/09/2019).

[19] M. Walker. (2005). Optimising the latency of your pc audio interface, [Online]. Available: https://www.soundonsound.com/techniques/optimising-latency-pc-audio-interface#7 (visited on 02/09/2019).

[20] T. M. Association. (2019). The midi manufacturers association (mma) and the association of music electronics industry (amei) announce midi 2.0™ prototyping, [Online]. Available: https://www.midi.org/articles-old/the-midi-manufacturers-association-mma-and-the-association-of-music-electronics-industry-amei-announce-midi-2-0tm-prototyping (visited on 05/26/2019).

[21] I. E. Sutherland, "The ultimate display," in *Proceedings of the IFIP Congress*, 1965, pp. 506–508.

[22] P. Cook, "Spasm: A real-time vocal tract physical model editor/controller and singer: The companion software synthesis system," *Computer Music Journal*, vol. 17, Jan. 1992. DOI: 10.2307/3680568.

[23] D. Trueman and P. Cook, "Bossa: The deconstructed violin reconstructed," *Journal of New Music Research*, vol. 29, Aug. 2010. DOI: 10.1076/jnmr.29.2.121.3098.

[24] V. Välimäki and T. Takala, "Virtual musical instruments - natural sound using physical models," *Organised Sound*, vol. 1, pp. 75–86, Aug. 1996. DOI: 10.1017/S1355771896000039.

[25] P. Cook, "Principles for designing computer music controllers," p. 4, Dec. 2001.

[26] T. Mäki-Patola, A. Kanerva, J. Laitinen, and T. Takala, "Experiments with virtual reality instruments.," Jan. 2005, pp. 11–16.

[27] R. Hamilton, "Building interactive networked musical environments using q3osc," Feb. 2009, p. 6.

[28] R. Hamilton. (2008). Quake iii open source control, [Online]. Available: https://ccrma.stanford.edu/~rob/q3osc/ (visited on 05/23/2019).

[29] M. Wright. (2002). Open sound control specification, [Online]. Available: http://opensoundcontrol.org/spec-1_0 (visited on 05/23/2019).

[30] R. Hamilton, "Maps and legends: Fps-based interfaces for composition and immersive performance," Jul. 2007, p. 4.

[31] S. Serafin, C. Erkut, J. Kojs, N. Nilsson, and R. Nordahl, "Virtual reality musical instruments: State of the art, design principles, and future directions," *Computer Music Journal*, vol. 40, pp. 22–40, Sep. 2016. DOI: 10.1162/COMJ_a_00372.

[32] G. Wang, "Principles of visual design for computer music," Sep. 2014, p. 6. DOI: 10.13140/2.1.3702.9763.

[33] R. Hamilton, "Udkosc: An immersive musical environment," Aug. 2011.

[34] S. Gelineck, N. Böttcher, L. Martinussen, and S. Serafin, "Virtual reality instruments capable of changing dimensions in real-time," May 2019.

[35] D. Bowman and L. Hodges, "An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments," *Symposium on Interactive 3D Graphics*, vol. 182, Sep. 1999. DOI: 10.1145/253284.253301.

[36] G. Levin, *Painterly inerfaces for audiovisual performance*, M.S. Thesis, MIT Media Laboratory, Jun. 2000.

[37] F. Berthaut, M. Desainte-Catherine, and M. Hachet, "Interacting with 3d reactive widgets for musical performance," *Journal of New Music Research*, vol. 40, pp. 253–263, Sep. 2011. DOI: 10.1080/09298215.2011.602693.

[38] J. Leonard, C. Cadoz, N. Castagné, and A. Luciani, "A virtual reality platform for musical creation," Oct. 2013. DOI: 10.1007/978-3-319-12976-1_22.

[39] N. Lobo, "V-drum: An augmented reality drum kit," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 10, Oct. 2015. DOI: 10.1080/09298215.2011.602693.

[40] *Microsoft kinect*, Aug. 2018. [Online]. Available: https://developer.microsoft.com/es-es/windows/kinect.

[41] K. Okada *et al.*, "Virtual drum: Ubiquitous and playful drum playing," pp. 419–421, Feb. 2015.

[42] H. Kanke, Y. Takegawa, T. Terada, and M. Tsukamoto, "Airstic drum: A drumstick for integration of real and virtual drums," vol. 7624, Nov. 2012, pp. 57–69. DOI: 10.1007/978-3-642-34292-9_5.

[43] J. Desnoyers-Stewart, D. Gerhard, and M. Smith, "Mixed reality midi keyboard," Sep. 2017.

[44] A. Fangbemi and Y. Zhang, "Wrist-worn sensor-based tangible interface for virtual percussion instruments," in. Jun. 2018, pp. 54–66. DOI: 10.1007/978-3-319-95270-3_4.

[45] *Wii music game details*, Aug. 2018. [Online]. Available: https://www.nintendo.com/games/detail/Fe0_TFVoa6RbkoZq_GoIDaRTgOzVAOID.

[46] COOLTHINGS. (2009). V-beat drumsticks: Imaginary drum kit, realistic playing, [Online]. Available: https://www.coolthings.com/v-beat-drumsticks-imaginary-drum-kit-realistic-playing/ (visited on 05/24/2019).

[47] *Our comprehensive aerodrums review*, Aug. 2018. [Online]. Available: https://www.electronicdrumadvisor.com/aerodrums-review/.

[48] *Aerodrums*, Aug. 2018. [Online]. Available: https://aerodrums.com/the-product/.

[49] *The music room*, Aug. 2018. [Online]. Available: https://www.vrfocus.com/2017/08/the-music-room-brings-real-instruments-to-vr/.

[50] S. Jordà, "Instruments and players: Some thoughts on digital lutherie," *Journal of New Music Research*, vol. 33, Sep. 2004. DOI: 10.1080/0929821042000317886.

[51] Thomann. (2019). Roland td-25k v-drum set, [Online]. Available: https://www.thomann.de/es/roland_td_25k_v_drum_set.htm (visited on 05/25/2019).

[52] J. Albano. (2017). Monitoring latency (how low can you go?) [Online]. Available: https://ask.audio/articles/monitoring-latency-how-low-can-you-go (visited on 05/26/2019).

[53] P. Cook, "Remutualizing the musical instrument: Co-design of synthesis algorithms and controllers," *Journal of New Music Research*, vol. 33, pp. 315–320, Sep. 2004. DOI: 10.1080/0929821042000317877.

[54] M. Collicutt, C. Casciato, and M. Wanderley, "From real to virtual: A comparison of input devices for percussion tasks," Jan. 2009.

[55] T. Mäki-Patola, "User interface comparison for virtual drums.," Jan. 2005, pp. 144–147.

[56] J. J. Gibson, *The ecological approach to visual perception*. Boston: Houghton Mifflin, 1979.

[57] S. Serafin, C. Erkut, J. Kojs, N. Nilsson, and R. Nordahl, "Virtual reality musical instruments: State of the art, design principles, and future directions," *Computer Music Journal*, vol. 40, pp. 22–40, Sep. 2016. DOI: 10.1162/COMJ_a_00372.

[58] B. Bache. (2019). A guide to drum sticks grip, [Online]. Available: https://www.libertyparkmusic.com/drum-sticks-grip-guide/ (visited on 05/26/2019).

[59] S. Nilsson, V. Vechev, A. Yeh, and C. Hedler, "Holo beats: Design and development of an ar system toteach drums," 2019.

[60] *Roland td-17 series*, Aug. 2018. [Online]. Available: https://www.roland.com/global/products/td-17_series.

[61] Thomann. (2019). Pearl p-530, [Online]. Available: https://www.thomann.de/es/pearl_p_530_bass_drum_pedal.htm (visited on 06/09/2019).

[62] ——, (2019). Tama hp30, [Online]. Available: https://www.thomann.de/es/tama_hp30_bass_drum_pedal.htm (visited on 06/09/2019).

[63] Reverb.com. (2019). Tama iron cobra jr., [Online]. Available: https://reverb.com/item/1762289-tama-iron-cobra-jr-double-bass-pedal (visited on 06/09/2019).

[64] Thomann. (2019). Bdp-s bass drum practice pad, [Online]. Available: https://www.thomann.de/es/millenium_bdp_s_bass_drum_practice_pad.htm (visited on 06/09/2019).

[65] ——, (2019). Evans rfbass practice pad, [Online]. Available: https://www.thomann.de/es/hq_percussion_rfbass_bass_pedal_practice_pad.htm (visited on 06/09/2019).

[66] ——, (2019). Meinl mpp-12-jb 12", [Online]. Available: https://www.thomann.de/es/meinl_mpp_12_jb_12_practice_pad.htm (visited on 06/09/2019).

[67] ——, (2019). Tama hs80w snare stand, [Online]. Available: https://www.thomann.de/es/tama_hs80w_snare_stand.htm (visited on 06/09/2019).

[68] M. Jones *et al.*, *Software Engineering Guides*. Hertfordshire, UK, UK: Prentice Hall International (UK) Ltd., 1996.

[69] Gracie. (2019). How much floor space do i need for a drum set? [Online]. Available: https://www.drummingbasics.com/average-floor-space-for-drums/ (visited on 06/09/2019).

[70] O. de Weck, *16.842 Fundamentals of Systems Engineering*. URL: https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-842-fundamentals-of-systems-engineering-fall-2015/# License: Creative Commons BY-NC-SA Last visited on 2019/05/01, 2015.

[71] Q. Company. (2019). About qt, [Online]. Available: https://wiki.qt.io/About_Qt (visited on 05/26/2019).

[72] K. Takahasi. (2019). Midi jack unity plugin, [Online]. Available: https://github.com/keijiro/MidiJack (visited on 05/26/2019).

[73] M. Gudino. (2017). Arduino uno vs. mega vs. micro, [Online]. Available: https://www.arrow.com/en/research-and-events/articles/arduino-uno-vs-mega-vs-micro (visited on 05/26/2019).

[74] H. ElHady. (2017). Top game engines, [Online]. Available: https://instabug.com/blog/game-engines/ (visited on 05/26/2019).

[75] P. Cook, "Re-designing principles for computer music controllers : A case study of squeezevox maggie," Jan. 2009.

[76] E. S. A. B. for Software Standardisation and Control, *Guide to Applying the ESA Software Engineering Standards to Small Software Projects*. European Space Agency, 1996. [Online]. Available: https://books.google.es/books?id=yy%5C_bHAAACAAJ.

[77] GeeksforGeeks. (2019). Software engineering — classical waterfall model, [Online]. Available: https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/ (visited on 06/09/2019).

[78] ——, (2019). Software engineering — spiral model, [Online]. Available: https://www.geeksforgeeks.org/software-engineering-spiral-model/ (visited on 06/09/2019).

[79] M. Rouse. (2019). Gantt chart definition, [Online]. Available: https://searchsoftwarequalit techtarget.com/definition/Gantt-chart (visited on 06/09/2019).

[80] A. Tributaria. (2018). Tabla de coeficientes de amortización lineal., [Online]. Available: https://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2015/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml (visited on 06/06/2019).

[81] D. O. de la Unión Europea. (2016). Reglamento general de protección de datos, [Online]. Available: https://www.boe.es/doue/2016/119/L00001-00088.pdf (visited on 06/09/2019).

[82] B. O. del Estado. (2018). Ley orgánica 3/2018, de 5 de diciembre, de protección de datos personales y garantía de los derechos digitales., [Online]. Available: https://www.boe.es/eli/es/lo/2018/12/05/3/dof/spa/pdf (visited on 06/09/2019).